UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

DETECCIÓN DE OBJETOS EN IMÁGENES DE HERRAJE INSTALADO EN EL TENDIDO DE FIBRA ÓPTICA DEL PROYECTO REGIONAL DE INSTALACIÓN DE BANDA ANCHA - PUNO 2020

TESIS

PRESENTADA POR:

Bach. WILBER MILTON MAYTA AROQUIPA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ESTADÍSTICO E INFORMÁTICO

PUNO – PERÚ

2021

DEDICATORIA

Con todo el amor y lealtad a mis queridos Padres: Jose Antonio y Salome quienes me inculcaron las buenas prácticas, la disciplina, respeto, responsabilidad y por todo el esfuerzo que pusieron en mi con paciencia y afecto, sacrificando infinidad de cosas para la culminación de mi carrera profesional.

A mi querida facultad de Ingenieria estadística e informática de la Universidad Nacional del Altiplano Puno, que llevare su nombre con mucho orgullo por donde vaya y que me dio la oportunidad de formarme profesionalmente.

Wilber M. Mayta A.

AGRADECIMIENTOS

Agradezco en primer lugar a Dios por darme la suficiente fortaleza y guiarme en el camino correcto, por cuidarme y darme la oportunidad de vivir en este mundo maravilloso durante mi formación profesional.

A mis hermanos Jhon y Fiorela quienes siempre estuvieron en los malos y buenos momentos para apoyarme en todo el tiempo de estudio para culminar satisfactoriamente.

A mi compañera, y gran persona Rosa Angélica que por su acompañamiento invaluable no hubiera sido posible consumar todo el resultado hasta la actualidad.

A mis amigos de infancia Victor Hugo, Fredy, Ivan P.,Ivan Ch. Abraham, Amidey, Rene A., Cesar , Alex, Dante, AbrahamL., entre otros por su apoyo incondicional que fortalecieron mi desarrollo personal.

A mis amigos de universidad que fueron los que acompañaron mi día a día de vida universitaria a Dennis Q., Ali, Brayan, Marvin, Waldir, Milton ,Rey Adan, DennisCH, Andy, Jhennery, DaysiM, Mary, Roxana y otros que confiaron, creyeron y formamos una gran amistad e influyeron para poder cumplir mis objetivos.

A los docentes en general con los que lleve los cursos en la escuela profesional de ingeniería estadística e informática por enseñarme todos sus conocimientos y compartir sus experiencias en mi formación profesional.

Wilber M. Mayta A.

ÍNDICE GENERAL

CAPÍTULO III

MATERIALES Y MÉTODOS

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

Área: Inteligencia Artificial.
Tema: Detección de objetos.

FECHA DE SUSTENTACIÓN: 04 de agosto de 2021

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ACRÓNIMOS

AP: Average Precision

Darknet: Marco de red neuronal de código abierto

FN: Falsos Negativos

FP: Falsos Positivos

GPU Graphics Processing Unit.

IA: Inteligencia Artificial.

IoU: Intersection over Union

JPEG: Joint fotographic expert Group

mAP: mean Average Precision

MTC : Ministerio de transportes y Comunicaciones

OpenCV: Open Source Computer Vision.

PRONATEL: Programa nacional de telecomunicaciones

Pytorch: Librería open source basada en Python

VN: Verdaderos Negativos

VP: Verdaderos Positivos

YOLO: You Only Look Once.(solo miras una vez)

RESUMEN

El campo de la visión computacional junto al deep learning de la inteligencia artificial, son áreas complementarias para el tratamiento de imágenes, y ello hizo posible desarrollar el presente trabajo de investigación, donde se tuvo por objetivo desarrollar algoritmos de detección de objetos en fotografías de herraje instalados en las estructuras del tramo Nuñoa – Macusani, con las últimas arquitecturas deep learning implementados hasta la actualidad. El primer paso fue la recopilación de fotografías para posteriormente etiquetar con el programa llabelling en el formato pascal Voc, luego se realizó la técnica de data Aumentation para tener la mayor cantidad de imágenes y de estos se separó en el 80% para entrenamiento y el 20% en test, posterior a ello se eligió la arquitectura Yolov4, y Yolov5. El alcance del estudio fue exploratorio, descriptivo; el diseño fue pre-experimental. Se desarrolló en Python en Google colab y con la técnica de transfer learning para YoloV4 se realizaron 6 pruebas diferentes donde se obtuvieron las mejores métricas luego de 14000 épocas de entrenamiento con batch size de 64 y el tiempo de entrenamiento de 16 horas aproximadamente, se logró obtener mAP de 89.45%, la precisión y recall que se obtuvo fue del 90%. Asimismo, para la arquitectura de YoloV5 se realizaron 7 pruebas diferentes donde se obtuvo las mejores métricas en 99 épocas de entrenamiento con batch size de 16 y el tiempo de entrenamiento de 4 horas aproximadamente logrando obtener mAP de 95.3% precisión de 95.3% y Recall de 94.5%. Finalmente se realizaron predicciones para validar el modelo obteniendo detecciones correctas de herraje instalado.

Palabras Clave: Deep Learning, Yolo, detección de objetos, herraje.

ABSTRACT

The field of computational vision together with deep learning of artificial intelligence, are complementary areas for the treatment of images, and this made it possible to develop the present research work, where the objective was to develop algorithms for the detection of objects in photographs of hardware. installed in the structures of the Nuñoa - Macusani section, with the latest deep learning architectures implemented to date. The first process that was carried out was the collection of photographs of each installed structure to later label with the llabelling program in the pascal Voc format, then the data Augmentation technique was carried out to have the largest number of images and these were separated in the 80% for training and 20% for testing, after which the Yolov4 and Yolov5 architecture were chosen. The scope of the study was descriptive, correlational and explanatory, the design was pre-experimental. The development of the code was in the Google platform collaborating with a synchronized drive account and with the transfer learning technique for YoloV4, 6 different tests were carried out where the best metrics were obtained after 14000 training periods with a batch size of 64 and the time of training of approximately 16 hours, it was possible to obtain a mAP of 89.45%, the precision and recall that was obtained was 90%. Likewise, for the YoloV5 architecture, 7 different tests were carried out where the best metrics were obtained in 99 training periods with a batch size of 16 and a training time of approximately 4 hours, obtaining mAP of 95.3%, precision of 95.3% and Recall of 94.5%. Finally, predictions were made to validate the model, obtaining correct detections of installed hardware.

Keywords: Deep Learning, computer vision, Yolo, object detection, chape.

CAPÍTULO I

INTRODUCCIÓN

Actualmente se viene ejecutando en la región de Puno el Proyecto regional de Banda Ancha que tiene como meta dotar de internet a todas las instituciones públicas existentes en los pueblos de la región de Puno desde los distritos, centros poblados y comunidades con población significativa dando prioridad a las instituciones educativas de nivel primario y secundario, puestos de salud y comisarias a nivel de toda la región de Puno.

Para este trabajo de investigación particularmente se tendrá en cuenta las estructuras de postes en el tendido de fibra óptica del tramo de Nuñoa-Macusani que tiene más de 142 kilómetros de tendido de fibra óptica y en donde se utiliza una variedad de partes de herraje instalados en los postes, en ese entender cada día de trabajo se emplean cantidades de gran volumen de herraje en cuanto a partes que se instalan en cada estructura que se le denomina ferretería o herrajeria. Entonces para contabilizar la cantidad de ferretería instalada en las estructuras es un problema ya que se tiene experiencia con otros tramos de tendido que se tuvo serias deficiencias al no tener un control adecuado en las cantidades de herraje que se utilizaron.

El problema que se apreció en anteriores tramos es que, para realizar la respectiva documentación y el conteo general de la ferretería utilizada, se cuenta con un personal exclusivamente para la recopilación de la información que consiste recopilar el registro fotográfico de cada estructura (llamado también en campo como ítem), donde el personal encargado para este trabajo registra fotografías desde diferentes ángulos.

Posterior a ello se realiza el levantamiento de información en un modelo de Excel y el primer problema que se identifica es el tiempo de demora en levantar dicha información donde se registra a partir de las fotografías los herrajes que contiene instalado cada poste y esto toma un tiempo determinado lo cual también se hace pesado para el que lo realiza porque es un trabajo mecánico de mirar las fotografías de cada estructura de poste y registrar en el archivo Excel.

Asimismo, otro problema identificado son los errores humanos, ya que se frecuenta cometer el error de tipeo, en el sentido de que en la foto se identifica un herraje determinado y al momento de registrarlo se apunta otra cosa.

También suele pasar la equivocación en el número de estructura (ítem) por ende se registra mal la información, porque se apunta todo lo verificado en la fotografía, pero al momento de regístralo en el archivo Excel se registró en otro ítem y este problema aparece en el momento de verificación de la información documentada con lo que está instalado en las estructuras que se tiene en campo ya que no coinciden.

Entonces por los problemas mencionados en los anteriores párrafos, es que se propuso implementar un modelo de detección de objetos utilizando las diferentes técnicas dentro del ámbito del machine learning, deep learning y la visión computacional y así poder automatizar los procesos y dar solución a los problemas detallados como la pérdida de tiempo, la equivocación de registrar el herraje correspondiente, y asimismo ahorrar en el personal para dicho trabajo; para ello será necesario encontrar un modelo optimo que pueda detectar todas las partes instaladas en los postes y que tenga los mínimos errores posibles.

1.1. FORMULACIÓN DEL PROBLEMA

¿Cuál es la precisión óptima que se obtiene con las arquitecturas de detección de objetos en herraje instalado en estructuras en el tendido de fibra óptica del tramo Nuñoa-Macusani en el Proyecto de Instalación de banda ancha?

1.2. JUSTIFICACIÓN DE LA INVESTIGACIÓN

En la actualidad con el avance de la inteligencia artificial, el machine learning y el ámbito de visión por computador donde se aplica el procesamiento de imágenes y algoritmos, se demostró que es posible realizar entrenamientos con arquitecturas modernas para la detección de objetos en imágenes, ya que tanto en hardware con la aparición de GPU´s y software como desarrollo de arquitecturas para manipulación de imágenes se vienen demostrando que es posible detectar objetos de diversos tipos con precisiones por encima del 90% lo cual es un avance significativo en este campo.

Entonces con la presente investigación se pretende encontrar la precisión más óptima, entrenando la arquitectura Yolo personalizado con diferentes parámetros, de detección de objetos mediante fotografías de los herrajes que tienen instalados los postes y esto contribuya significativamente a la empresa optimizando tiempo y error de tipeo del levantamiento de información de ferretería instalada en cada estructura por el personal encargado.

También con la presente investigación se pretende contribuir en este campo y demostrar que la inteligencia artificial en su área de aprendizaje automático y visión por computador se pueda aplicar en cualquier campo o área de trabajo, teniendo en cuenta que este tipo de estudios aun es poco conocido y aplicado ya sea por desconocimiento o porque no existe bastante información o que la bibliografía está en el idioma ingles en

su gran mayoría por ser algo reciente. Entonces el presente proyecto espera apoyar en presentes y futuras investigaciones en este ámbito de estudio y motivar el desarrollo de estos temas.

1.3. HIPÓTESIS DE LA INVESTIGACIÓN

Utilizando diferentes parámetros en las arquitecturas de detección de objetos con redes neuronales profundas, es posible encontrar un algoritmo óptimo para la detección de tipos de herraje instalado en las estructuras del tramo Nuñoa – Macusani del Proyecto de Instalación de banda ancha en la región de Puno.

1.4. OBJETIVOS DE LA INVESTIGACIÓN

1.4.1. Objetivo general

Desarrollar algoritmos de detección de objetos en fotografías de herraje instalada en las estructuras del tramo Nuñoa – Macusani en el Proyecto de Instalación de banda ancha Puno.

1.4.2. Objetivos específicos

Examinar diferentes métricas de evaluación en las arquitecturas implementadas de aprendizaje profundo para la detección de objetos en imágenes.

Identificar la arquitectura con mejores métricas de evaluación implementado para la detección de objetos en fotografías de herraje instalada en las estructuras del tramo Nuñoa – Macusani en el Proyecto de Instalación de banda ancha.

Describir los tipos y partes de herrajes instaladas en las estructuras del tramo Nuñoa – Macusani que se detectaron mediante las arquitecturas implementadas con mejores métricas de evaluación.

CAPÍTULO II

REVISIÓN DE LITERATURA

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

2.1.1. Nivel internacional

(Castro, 2020) en su tesis denominado DETECCIÓN Y CLASIFICACIÓN DE CÉLULAS NORMALES DE LA SANGRE PERIFÉRICA USANDO APRENDIZAJE PROFUNDO donde entreno modelos con redes neuronales convolucionales (CNN) y uso arquitecturas de CNN para la obtención de características de forma automática; logro entrenar una red desde cero para el clasificador con un accuracy de 0.862 y para el detector utilizo la arquitectura YOLO obteniendo un excelente desempeño.

(Gutierrez, 2019) en su tesis titulada DETECCIÓN DE ARMAS EN VÍDEOS MEDIANTE TÉCNICAS DE DEEP LEARNING cuyo objetivo fue crear un sistema capaz de detectar armas de forma automática en imágenes y vídeos; el principal algoritmo que estudio y optimizo fue con Yolo “You only looks once”, con el 71% de mAP, además de ellos uso los algoritmos SSD300 y SSD512, donde concluye que el tiempo de predicción Yolo v2 es el ganador absoluto consiguiendo ser aproximadamente 10 veces más veloz que el SSD300.

(Garralda, 2018) en su tesis titulada LOCALIZACIÓN DE OBJETOS EN IMÁGENES MEDIANTE TÉCNICAS DE APRENDIZAJE PROFUNDO donde su objetivo es crear un modelo utilizando técnicas de aprendizaje profundo que sea capaz de determinar la posición de un violín en una imagen, ella lo realiza con el fin de contribuir para futuros

proyectos de su universidad donde concluye que su modelo optimo fue menos del 65% e indica que en problemas de localización es difícil obtener una buena precisión.

(Arriola, 2018) en su trabajo final de master titulado DETECCIÓN DE OBJETOS BASADA EN DEEP LEARNING Y APLICADA A VEHÍCULOS AUTÓNOMOS el trabajo lo desarrollo en el contexto de los vehículos autónomos para lo cual entreno y comparo tres redes Faster-RCNN para la detección de peatones partiendo desde diferentes inicializaciones de sus parámetros para estudiar la influencia de la transferencia del aprendizaje. Desarrollo utilizando la librería Tensorflow de google llegando a las conclusiones de que el modelo Faster-RCNN Resnet101 alcanza valores del mAP del 82 %, mientras que el Faster-RCNN Inception v2 toma un valor de mAP del 75% a un costo de procesamiento más bajo.

2.1.2. **Nivel nacional**

(Lazo, 2019) en su trabajo de tesis titulado “ESPECTROSCOPIA CON INFRARROJO Y TÉCNICAS DE MACHINE LEARNING Y DEEP LEARNING PARA LA DETECCIÓN Y CLASIFICACIÓN DE FRUTAS PARA LA AGROINDUSTRIA: CASO ARÁNDANOS EMPRESA TAISA:2018”, que tiene como finalidad establecer una herramienta alternativa para poder hacer la clasificación de arándanos que permita reducir el costo el tiempo y optimizar el proceso de detección y clasificación de frutas utilizando la librería keras con tensorflow, encontró un modelo que clasifique los arándanos con una exactitud del 92%.

(Huaman, 2019) en su trabajo de investigación titulado “DETECCIÓN DE ARMAS DE FUEGO UTILIZANDO REDES CONVOLUCIONALES PROFUNDAS” donde el objetivo de su trabajo fue identificar armas de fuego en videos y para ello reviso el

estado de arte de la red Faster- R –CNN identificado solo a una velocidad de 7fps, seguidamente entreno en la arquitectura YOLO teniendo resultados de 45 fps con una tasa de acierto del 97.30%.

(Gutierrez, 2019) en su tesis titulado "RECONOCIMIENTO DE EVENTOS ANÓMALOS EN VIDEOS OBTENIDOS DE CÁMARAS DE VIGILANCIA, USANDO REDES CONVOLUCIONALES" tiene como objetivo detectar eventos anómalos en los videos de vigilancia y que no haya la necesidad de que intervenga alguna persona encargada para verificar dichos actos anómalos, para ello utiliza las arquitecturas de YOLO, Región Proposal + CNN(R-CNN) y el SSD encontrando al final un acierto del 86% de accuracy, 85% de precisión y 73% de recall en los videos además de ello indicando que YOLO es la arquitectura que procesa más rápido los videos en tiempo real.

2.1.3. **Nivel regional**

(Chino, 2019) en su tesis denominado DISEÑO DE UN ALGORITMO DE PROCESAMIENTO DE IMÁGENES DEL SISTEMA DE PESAJE PARA EL CONTROL AUTOMÁTICO DE UNA FAJA TRANSPORTADORA EN LA UNIDAD MINERA MALLAY, teniendo como objetivo Diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora y reducir la parada de producción donde concluye que el pesaje en el algoritmo presenta una precisión del 99% y margen de error del 1%, que es aceptable de acuerdo a los objetivos de su investigación, él recomienda OPenCV como una herramienta efectivo para este tipo de investigaciones.

(Ccari, 2019) en su tesis DETECCIÓN DE EVENTOS INUSUALES EN IMÁGENES Y VIDEO DE CÁMARAS DE VIGILANCIA se propone desarrollar y utilizar los algoritmos y las técnicas para la detección de eventos y acciones inusuales en humanos y para luego probarlo en videos por medio de las cámaras de vigilancia. Para dicha aplicación él hace uso del algoritmo y técnica de "máquinas de postura convolucional", donde en la primera fase realiza el uso de los campos de afinidad de las partes del cuerpo, para encontrar los puntos de interés (articulaciones) luego aplica una red neuronal convolucional concluyendo que crear un sistema computacional capaz de procesar imágenes en video y detectar acciones y/o eventos inusuales en movimiento de personas es posible con las técnicas mostradas en ese trabajo.

(Ordoñes, 2020) en su tesis de nombre DEEP LEARNING PARA LA VISIÓN ARTIFICIAL E IDENTIFICACIÓN DEL PERSONAL ADMINISTRATIVO Y DOCENTE DE LA UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC 2018 donde se propuso cm objetivo lograr la proporción más alta de precisión en la identificación del personal administrativo y docente y que tuvo como resultados dos modelos principales y eficientes de los cuales el primero fue el modelo VGG16UNAMBA con el cual logró obtener una proporción de 0.9805 de precisión; mientras que el segundo modelo lo denomino DenseNet121UNAMBA,en donde logró una proporción de 0.9932 de precisión.

2.2. MARCO TEÓRICO

2.2.1. **Marco conceptual**

En este capítulo se efectúa una revisión literaria sobre conceptos de la teoría que se basa la aplicación de la investigación que vienen de temas de redes neuronales

convolucionales, inteligencia artificial, redes neuronales, visión por computadora, principales arquitecturas de la detección de imágenes que existen hasta la actualidad, estudio a detalle de la arquitectura YOLO que está basado la presente tesis, y todos estos temas serán de gran uso para el desarrollo y la detección de herraje instalado en las estructuras de los postes del tramo Nuñoa - Macusani.

2.2.2. **Inicialización del modelo**

Para empezar a detectar un modelo de visión computacional, es necesario saber la teoría de que la persona humana capacite, entrene, enseñe al modelo el concepto inicial del rastreo de un objeto a través del color, textura, separado por categorías entre otros (Hernández García et al., 2017)

El procedimiento para identificar cual es la inicialización del movimiento cinemáticos del cuerpo humano desde la parte superior del cuerpo es utilizando la técnica de la segmentación en videos monoculares.

2.2.3. **Inteligencia artificial**

La inteligencia artificial viene en desarrollo desde antes de los años noventa, y es parte de las ciencias de la computación y se empezó a aplicar las teorías existentes a partir de los sistemas computacionales donde estos sistemas desarrollan automáticamente las tareas que una persona humana los desarrollaría, como reconocimiento de imágenes, clasificación, detección, toma de decisiones, entre otros.(Sanlam, 2018)

La inteligencia artificial (IA) es una de las ramas de la Informática, con fuertes raíces en otras áreas como la lógica y las ciencias cognitivas. Por lo tanto la IA se basa en cuatro principios importantes los cuales son: actuar como personas, razonar como personas,

razonar racionalmente, actuar racionalmente y todas estas definiciones se pueden aplicar en el lenguaje natural, la visión artificial, la robótica, entre otros.(Torra, 2020)

Figura 1. Subcampos de la inteligencia artificial
Fuente: (Sanlam, 2018)

2.2.3.1. **Temas fundamentales de la ia**

Según (Ponce, 2010) el campo de la IA se compone de varias áreas de estudio, las más comunes e importantes son:

- Incertidumbre y "lógica difusa"
- Redes neuronales artificiales
- Algoritmos genéticos

La presente investigación está basada en uno de los campos de la inteligencia artificial cual es el aprendizaje automático de máquinas o conocido en el ámbito de la ciencia de datos como machine learning.

2.2.4. **Redes neuronales**

El estudio de las redes neuronales artificiales está basado en la funcionalidad del sistema nervioso humano. La red neuronal artificial está formada por un sistema de neuronas que vienen a estar interconectados entre sí por otras neuronas denominados enlaces, cada neurona adquiere entrada de lo que sale de la neurona antecesora, una vez

que entra una información a una neurona esta información se multiplica por un peso determinado manualmente y posterior sistemáticamente, en seguida se realiza una función de activación, para luego llegar a calcular una salida que posteriormente será la entrada a otra neurona; el mismo proceso vuelve a repetirse hasta que pase todas las capas de la red neuronal. Todas estas neuronas interconectadas y el proceso que se realiza en cada capa es lo que se llama red neuronal artificial. Estas redes son interconectadas masivamente en paralelo de elementos simples (usualmente adaptativo) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico (Calvo, 2018)

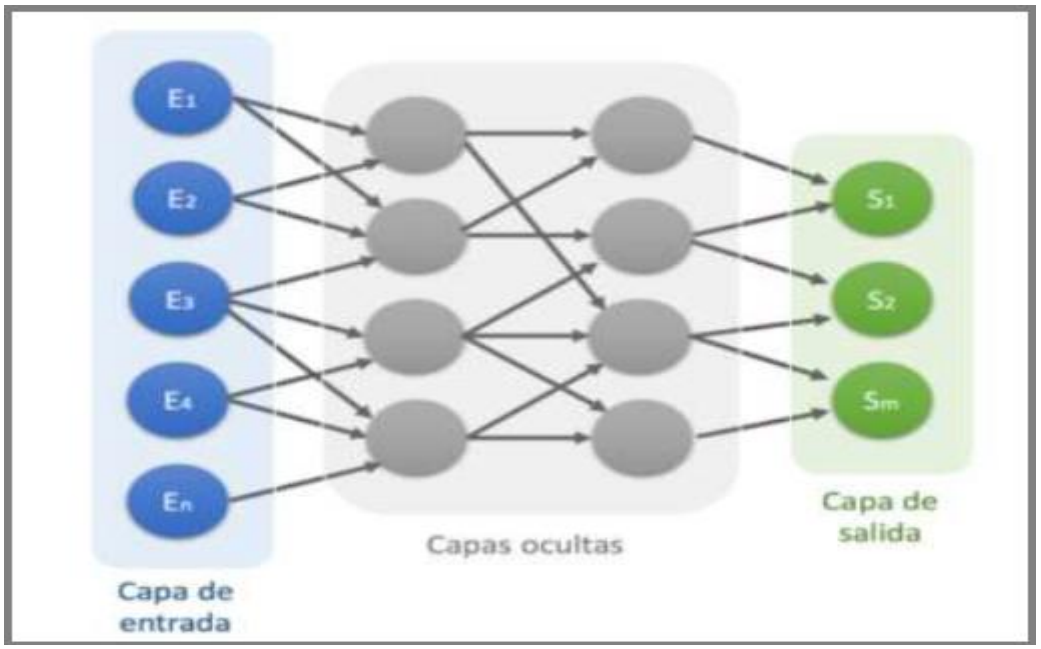


Figura 2. Red neuronal artificial
Fuente: (Calvo, 2018)

2.2.5. **Redes neuronales artificiales**

Las redes neuronales artificiales se basan en una inspiración biológica de la red neuronal humana, y a partir de diferentes métodos y algoritmos hace que sea automático y a la vez vaya aprendiendo mientras se le asigne una cantidad de conjunto de datos. La

red neuronal artificial tiene partes como las capas y dentro de cada capa existen neuronas que se procesan en capa y al final logran generar en la última capa una salida de por ejemplo 0 y 1, la manera en cómo aprenden las redes neuronales artificiales es mediante el error que se comete al realizar un paso general en toda la capa, es decir, al inicio se tiene un conjunto de datos, seguidamente estos pasan a la primera capa y en aquí se realizan operaciones con el peso determinado aleatoriamente y una función de activación que se le asigne y así al pasar por todas las capas, al final llega a predecir si va a ser un 0 o 1, entonces esto se compara con el verdadero valor que tenga dicha información y si la predicción es correcta para ese dato, los pesos ya se estarían ajustando, caso contrario vuelve a retroceder por las capas que paso, pero ajustando el peso aleatorio que se le asignó a un inicio, esto se realiza hasta ajustar los pesos simbolizados como "Wi", mientras se tenga gran cantidad de conjuntos de datos y se vaya pasando una otra vez por la red neuronal para ajustar los pesos, llegara un momento donde se reduzca gradualmente el error en general y así se formara un modelo (Martínez, 2018).

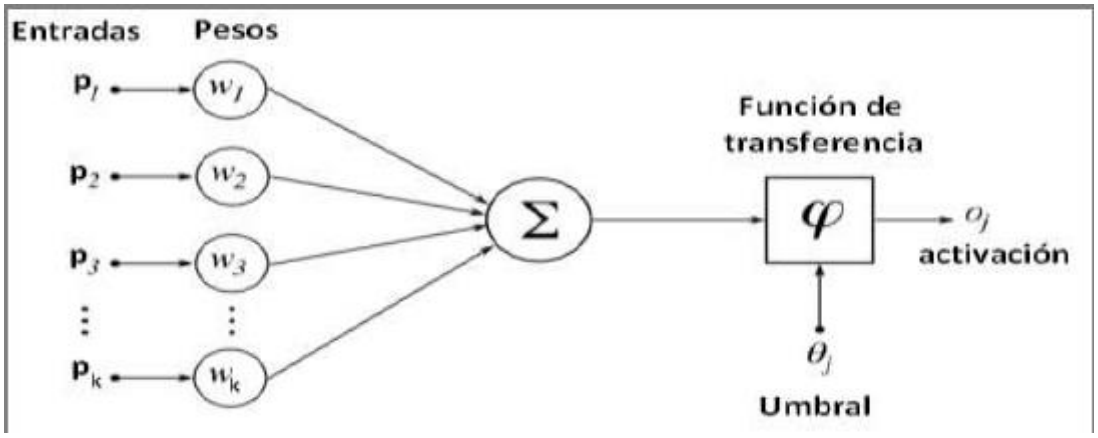


Figura 3. Diagrama de una neurona con la función de activación umbral
Fuente: (Kawaguchi, 2000)

2.2.6. **Machine learning**

Es una rama de la inteligencia artificial (IA) que hace que los programas adaptados a diferentes dispositivos electrónicos mediante técnicas y/o algoritmos desarrollados tengan la suficiente capacidad de aprender y mejorar sin que sean programadas automáticamente. Este técnica de aprendizaje supervisado se desarrolla con la ayuda de la informática y la teoría de la ciencia de datos es así que cuando se les asigne por primera vez un dato entonces tenga la suficiente capacidad de decidir a qué clasificación le corresponda(Raschka & Mirjalili, 2019)

Enfocándonos en el Machine Learning constan de tres diferentes técnicas para aprender que se detallaran a continuación y son los siguientes, el aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por reforzamiento (Hurwitz & Kirsch, 2018).

2.2.6.1. **Aprendizaje supervisado**

El aprendizaje supervisado es la técnica más conocida y de mayor utilización y se desarrolla a partir de datos etiquetados esto quiere decir que al encontrar un conjunto de datos estos mismos deben de tener su clasificación para que mediante ellas se pueda ver en cuanto está prediciendo correctamente, por ejemplo si se le entrega un conjunto de datos de dos tipos de animales como vaca y oveja, cada imagen de ellas debe estar correctamente etiquetado si es vaca u oveja, entonces nosotros le vamos mostrando al sistema imagen por imagen y al mismo tiempo indicándole que tipo de animal es para que así vaya aprendiendo a través de sus características que tenga cada imagen como tamaño, color, forma entre otros. Finalmente una vez que la red haya sido entrenada con una suficiente cantidad de datos, cuando se le indique clasificar una imagen que no haya

sido entrenada esta podrá identificar si la imagen corresponda a una vaca o una oveja (Hurwitz & Kirsch, 2018)

En esta investigación se utiliza el aprendizaje supervisado.

2.2.7. **Deep learning**

Es una rama del machine learning que se aplica fundamentalmente para el estudio de imágenes, se define como un conjunto de técnicas y pasos a seguir formando un algoritmo para lograr que un dispositivo aprenda de la misma manera que haría el trabajo un ser humano. Existen una diversidad de conjuntos de algoritmos, todas ellas se basan en un objetivo cual es simular el comportamiento del cerebro biológico pensante, y ponerlo en práctica para el reconocimiento de imágenes, identificación de sonidos y la comprensión de un idioma o lenguaje. Estos algoritmos son tan complejos porque funcionan con diversas cantidades de capas en una red neuronal artificial simulando cómo funciona el cerebro humano a través de las neuronas (García, 2015).

2.2.7.1. **Principales algoritmos del deep learning**

Los tipos de deep learning de acuerdo a su arquitectura y a su finalidad que persiguen los clasifican en tres grupos (Deng & Yu, 2013):

1) Redes profundas para aprendizaje no supervisado o generativo

En estas redes, el propósito principal es el análisis de patrones, la síntesis de los datos observados, o bien una agrupación, sin que se tenga una etiqueta para cada clase de patrón u objetivo.

2) Redes profundas para aprendizaje supervisado

En estas redes, se cuenta con patrones conocidos y bien categorizados, con el fin de clasificar de forma directa.

3) Redes profundas híbridas

Es una mezcla de las anteriores, con la meta de poder tener la capacidad de discriminar, auxiliado del aprendizaje no supervisado. Y esto podría llevar a un mejor performance que las redes supervisadas.

2.2.8. Redes neuronales convolucionales

Las redes neuronales convolucionales son una variante de las redes neuronales artificiales, lo que les diferencia es que se realizan convoluciones entre los parámetros y los datos de la red, estos tipos de modelos son apropiados para aquellos datos que se tengan en forma de rejillas o matrices, es por ello que se aplican mayormente a imágenes ya que estas vienen determinados por matrices y se procesan por secciones y ello están teniendo un notable éxito (Vizcaya et al., 2017)

Las redes neuronales convolucionales (CNNs) son de los algoritmos más populares en el paradigma del Deep Learning, cuando se trata de reconocimiento de objetos en imágenes. El funcionamiento que han tenido hasta la actualidad les ubica entre los mejores en comparación con otros tipos de modelos para reconocimiento de imágenes. Su teoría se basa en las primeras investigaciones realizadas de Lecun en el año de 1998, en ese entonces solo se tenía como teoría ya que no se contaba con el hardware adecuado para la suficiente capacidad de procesamiento de imágenes, es por ello que en el año 2011 con la aparición de las GPU’s se logra tener un avance importante cuando

se obtuvieron buenos resultados en determinar la clasificación de objetos y en otro concurso donde se tenía por objetivo reconocer dígitos escritos a mano(Nielsen, 2019)

Desde ese entonces se empezaron a tomar mayor importancia en las redes neuronales convolucionales, se empezaron a investigar con otras variantes, desarrollaron modelos de arquitecturas diferentes; de las principales compañías dedicadas a este rubro.

Las redes neuronales convolucionales están inspirados en el área del cerebro, para ser más precisos en el área de la corteza visual, es esta parte del cerebro biológico la que se encarga de recibir la información visual y tiene diversas funcionalidades para procesar lo que observa ya que contiene cantidad de células que recepciona y detectan la luz, asimismo estas células estas sujetas a la operación de convolución ya que todas las células procesan su entrada de la misma manera (Vizcaya et al., 2017).

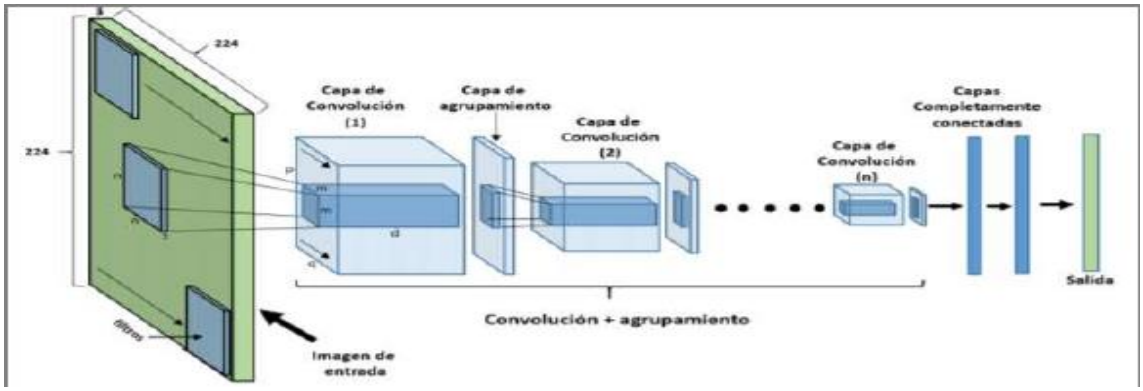


Figura 4. Arquitectura básica de una red neuronal convolucional
Fuente: (Vizcaya et al., 2017)

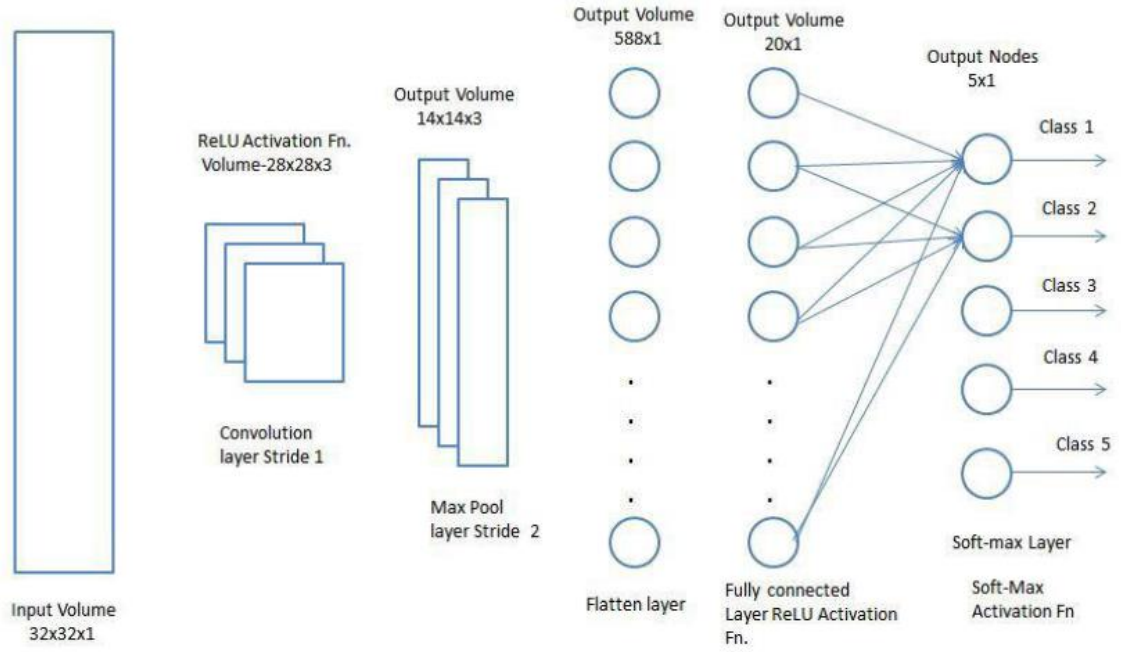


Figura 5. Partes de una red neuronal convolucional

2.2.9. **Capa de convolucionales**

El proceso de convolución para cada capa, es fundamental por ello en primer lugar se debe tener en cuenta que los pesos que tenga una capa sean compartidos en toda esa misma capa; esto quiere indicarnos; que los filtros aplicados en una capa son únicos en toda esa capa, ya que ese mismo filtro va a ir recorriendo a través de todo el conjunto de datos de entrada. Esto hace que al final se tenga un resultado denominado como mapa de activación bidimensional, y es conocido como mapa de características (Vizcaya et al., 2017)

El filtro en sí está sujeto a cuatro hiperparámetros:

- El tamaño (size)
- La profundidad (depth)
- La zancada (stride) y
- El relleno de ceros (zero-padding).

$$S(i.j) = (I * K)(i,f) = \sum_m \sum_n I(m,n)K(i-m,j-n)$$

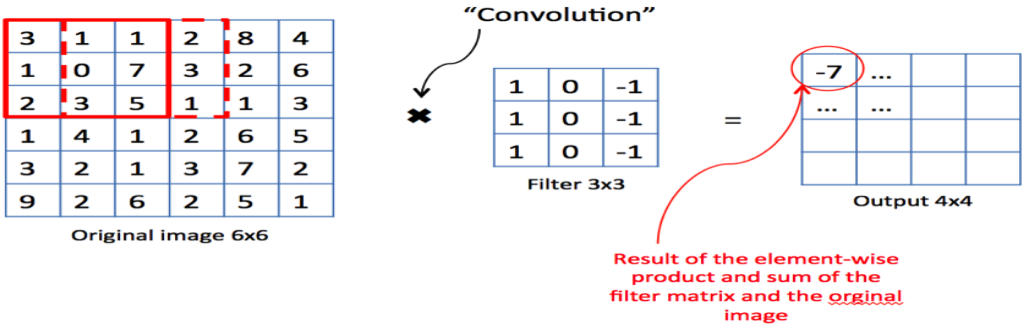


Figura 6. Capa de la convolución
Fuente: (Rodríguez, 2018)

2.2.9.1. **El tamaño (size)**

El tamaño de una convolución viene dada por el área de filtro que a estar avanzando la imagen, generalmente un filtro es cuadrado pero no siempre será así, los filtros más grandes tenderán a superponerse más, esto puede mejorar la precisión de la clasificación; sin embargo, aumentar el tamaño del filtro creará salidas cada vez más grandes; por lo que, administrar el tamaño de los resultados de las capas convolucionales es un factor importante para controlar la eficiencia de una red (Erroz, 2019).

2.2.9.2. **La profundidad (depth)**

La profundidad viene especificada como el número de nodos que tendrá la capa. También se le conoce como dimensión de una red, está estrechamente relacionado con la complejidad de la imagen dada, esto viene comprendido por la cantidad de canales necesarios para describirla con precisión, para determinar los parámetros de profundidad tiene que ser expresiva; ya que si establecemos parámetros con profundidad pequeña se puede tener malos resultados. Por ello para lograr la cantidad

suficiente de parámetros en términos de profundidad en una red se realiza mediante la optimización de hiperparámetros (Erroz, 2019).

2.2.9.3. **La zancada (stride)**

El stride es el salto que va a dar cada filtro en la imagen, es decir un salto o recorrido de uno en uno se hará que cada pixel de la imagen sea el centro de una instancia de filtro y esto conduce a que se tengan grandes resultados en el mapa de características, ahora por el contrario cuando el stride sea muy grande se va causar de que el mapa de características sea pequeño y por ende se reduce rápidamente el tamaño de la imagen inicial (Erroz, 2019).

El siguiente diagrama muestra gráficamente tanto la profundidad como la zancada

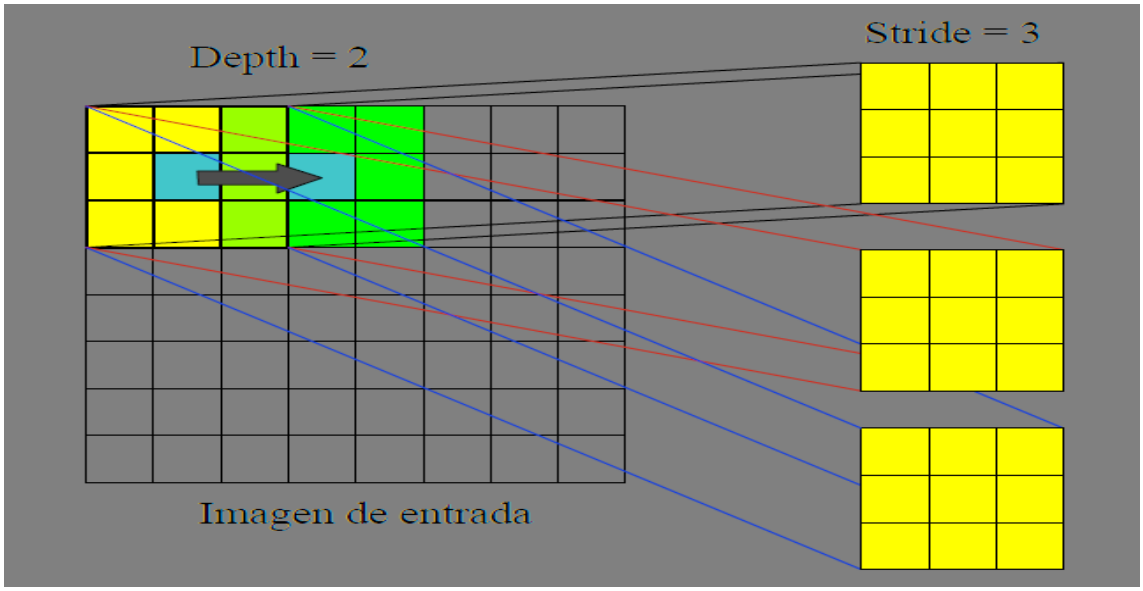


Figura 7. Depth and stride
Fuente: (Erroz, 2019)

2.2.9.4. **El relleno cero (zero padding),**

Dependiendo de los anteriores parámetros, se percibirá si los bordes de la imagen quedaran libres es decir que no se les pueda aplicar la convolución, entonces para ello, se aplica esta técnica del relleno con ceros, también se puede establecer alrededor del

borde de la imagen con ceros, esto hace que se disminuya rápidamente el tamaño de la imagen pero en consecuencia también se puede perder la efectividad ya que pueda ser que en los bordes se tenga información importante, por lo tanto, incrementar ceros pueda causar disminución de precisión pero será necesario aplicarlo cuando la convolución lo requiera (Erroz, 2019)

2.2.10.**Capas de agrupamiento o pooling**

La principal función de esta capa es la de reducir el tamaño de la imagen a medida que se vaya realizando la convolución, así se irá reduciendo los parámetros y por ende también se reducirá considerablemente la cantidad de cálculos a realizar, para ello generalmente se aplica la técnica del máximo valor de agrupamiento o conocido como maxpooling que consiste en subdividir en partes iguales y tomar el máximo valor de dicha subdivisión, eliminado los valores que tengan el mínimo en dicha subdivisión (Vizcaya, 2017).

Si no agrupamos, tendemos a encontrarnos con una gran cantidad de características; por lo cual, su uso disminuye el tamaño con una menor sobrecarga de cálculo para las próximas capas de la red; así como también, reduce el sobre ajuste (overfitting).

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1$$

$$H_2 = \frac{(\mathrm{H}_1 - F + 2P)}{S} + 1$$

$$D_2 = K$$

2.2.11.**Capa completamente conectada**

Como indica el nombre, cada nodo está conectado con cada una de las salidas de la capa que le antecede, Cada nodo realiza una suma ponderada de sus parámetros, esto con el fin de que tenga la máxima puntuación, y así el nodo que obtenga mayor puntuación será el que tenga también la mayor probabilidad de ocurrencia, el problema que se pudiera ver con esta etapa es que se tendría que realizar demasiados cálculos por ello lo recomendable es quitar los enlaces aleatoriamente para que no tenga bastante calculo computacional. (Vizcaya et al., 2017)

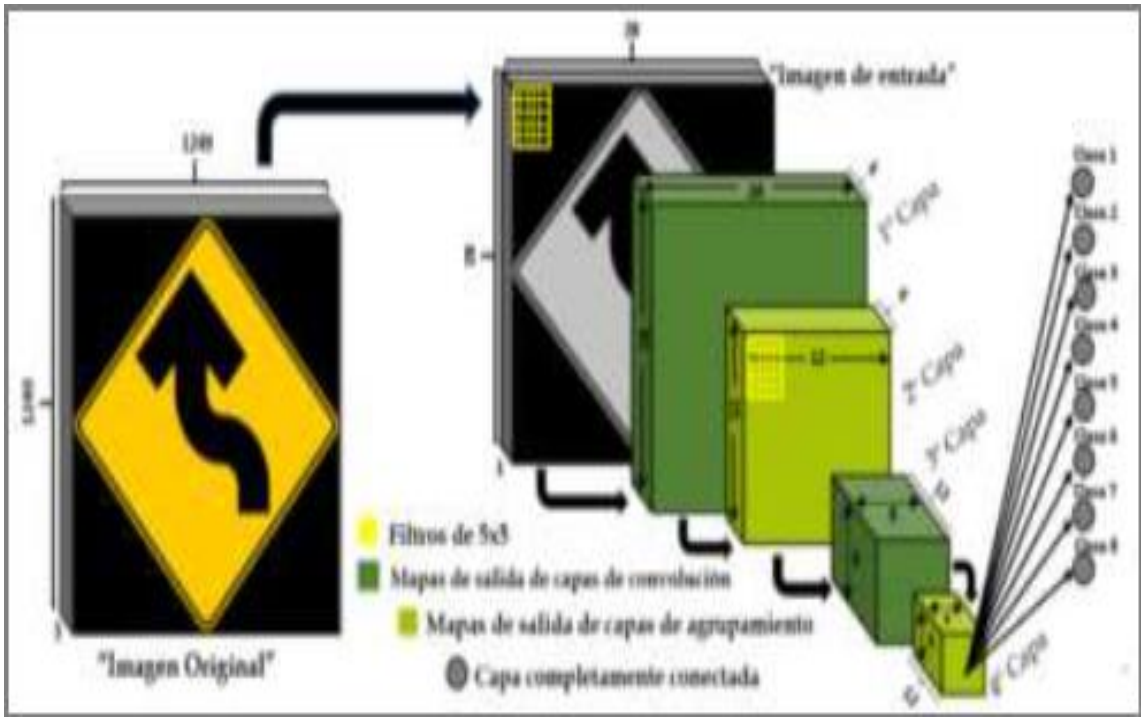


Figura 8. Arquitectura implementada completamente conectada
Fuente: (Vizcaya et al., 2017)

2.2.12.**Funciones de activación**

La función de activación es la que define la salida de la neurona en términos de la entrada dada y se realiza mediante transformaciones no lineales de las cuales la más utilizada es la función Relu (Erroz, 2019).

Sigmoid

$$f(x) = \frac{2}{1 + 2^{-2x} - 1}$$

Relu:

$$f(x) = max\ (0, x) = \begin{Bmatrix} 0\ for\ x < 0 \\ x\ for\ x \geq 0 \end{Bmatrix}$$

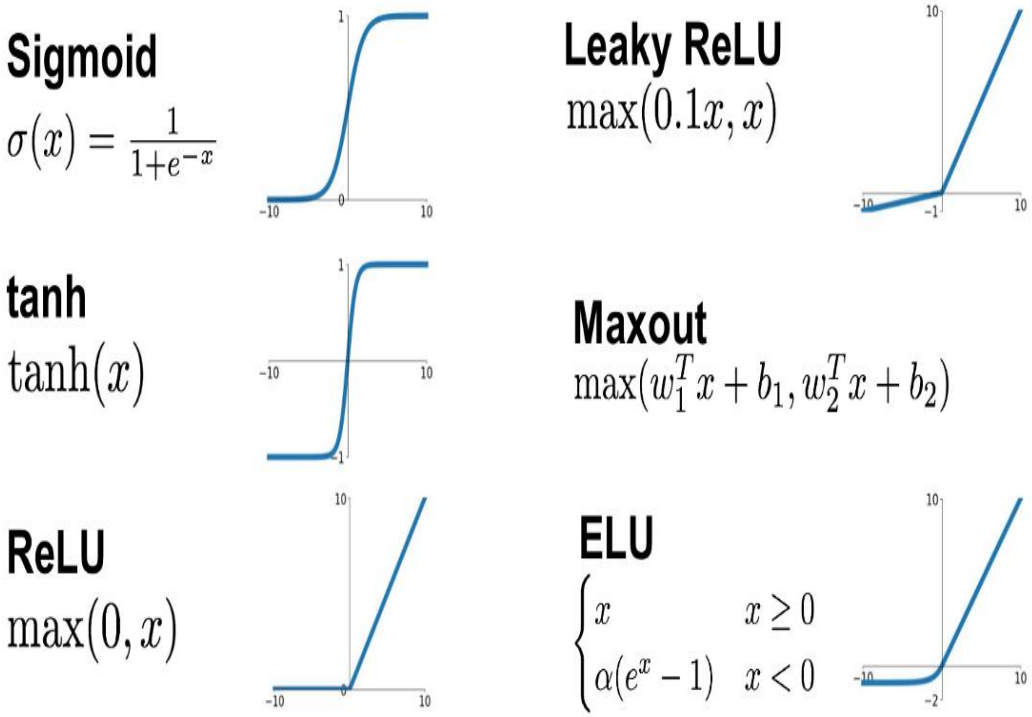


Figura 9. Funciones de activación utilizadas en deep learning

2.2.13.**Entrenamiento de redes neuronales convolucionales**

2.2.13.1. **Gradient descent:**

Es la función de coste que más se aplica en la actualidad, esta función lo que determina es minimizar la función de coste para ello se analiza durante cada época de entrenamiento y se va modificando los pesos a través de derivadas en cada capa donde

nos indique la dirección que debemos ir para modificar los pesos si es incrementar o disminuir de acuerdo al resultado matemático que se obtenga(Erroz, 2019)

2.2.13.2. **Batch size:**

Es el tamaño de lote o número de muestras que se va a propagar hacia adelante y hacia atrás en la red. Como ejemplo si se tiene 1000 muestras de imágenes y el bath size es de 100, se cogerá cada 100 imágenes para ir modificando los pesos a través del descenso de gradientes u otra función de coste. Dependiendo de la capacidad de memoria RAM puede que al indicar como parámetro un batch size de mayor numero se produzca colapso de memoria es por ello que en algunos modelos se pueda utilizar minibatch para que haya óptimo rendimiento de memoria RAM y hardware. (Erroz, 2019)

2.2.13.3. **Epoch:**

época en español que quiere decir el número de veces que se tomara todo el conjunto de datos de entrenamiento. Siguiendo el ejemplo anterior, tomando cada 100 imágenes por muestra y al tener 1000 imágenes, entonces se requerirá 10 ciclos para haber pasado por todas las imágenes y todo esto será una época, posterior a ello se volverá a realizar el mismo procedimiento dependiendo de la cantidad de épocas que se proponga. (Erroz, 2019)

2.2.13.4. **Learning rate:**

Este parámetro es lo que indica la cantidad de cambio que habrá al modificar la función de coste, es decir si la función de coste indica incrementar el peso, entonces el learning rate indica en cuanto se va a incrementar ese peso, ya que si se coloca como parámetro un número grande entonces los saltos serán enormes y no se lograra encontrar el modelo optimo pero la ventaja es que se pueda encontrar rápidamente el punto de precisión

máxima pero esto no sucede generalmente por ello el parámetro recomendable es de 0.001 (Erroz, 2019)

2.2.13.5. **Batch normalitation:**

El batch normalization es una técnica de la normalización estándar de datos, es decir la salida que tenga como resultado cada capa, se le resta el promedio del batch y ese resultado se divide entre la desviación del batch; esto se hace con la condición de que los rangos de diferencias sean menores y el aprendizaje sea más efectivo y rápido y por ende el learning rate podría utilizarse valores un poco más altos ya que el rango de variación disminuirá. (Erroz, 2019)

2.2.13.6. **Sgd y adam**

El descenso de gradientes estocástico es el que se aplica a una sola imagen en toda la arquitectura va de avance como de regreso en el modelo, es decir vendría a ser un mini batch de 1, ahora Adam es otra función de optimización del modelo donde toma la imagen de uno en uno, pero con la diferencia de que el learning rate puede variar en cualquier momento y por ende es más efectivo y el más utilizado en aprendizaje profundo.

2.2.13.7. **Sobre ajuste y subaajuste**

Conocidos como overfitting y underfitting y esto es para que el modelo no esté muy ajustado a los datos o viceversa que falte ajustarse a los datos de entrenamiento y esto se comprueba mediante la validación que se realiza donde son predicciones realizadas en imágenes que no fueron utilizadas en el entrenamiento ni en el test. El overfitting ocurre cuando las imágenes de entrenamiento tienes características que no contribuyen al modelo, y por ende este aprenderá esas características que no son comunes como ruido,

distorsiones, y el modelo solo predecirá los datos que son parecidos e idénticos a los datos de entrenamiento. El otro caso del Underfitting es lo contrario cuando la red no es entrenada con suficientes datos y de diversos tipos, ángulos, ya que así no se generaliza y por eso el modelo es incapaz de detectar correctamente en datos nuevos. (Erroz, 2019)

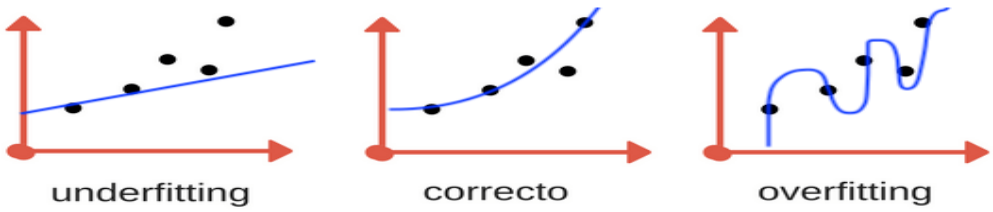


Figura 10. Tipos de ajuste que se pueden presentar en un modelo.

2.2.14.**Evaluación de la efectividad de un modelo de deep learning**

Al momento de evaluar el rendimiento que tenga nuestro modelo se toma generalmente la matriz de confusión, esta matriz viene a ser una tabla bidimensional de contingencias, como se conoce en el ámbito estadístico, donde se detalla en las filas las etiquetas que predijeron el modelo versus las columnas que son las etiquetas verdaderas que tiene cada dato. Por ende, en esta matriz de confusión se podrá notar la cantidad de datos que fueron predichos correctamente que estarán en las diagonales de dicha tabla de contingencia, cuando se tiene más de 2 categorías a predecir, es muy recomendable realizar este tipo de matriz de confusión, ya que se evidenciará información sobre los errores en la clasificación y así poder evaluar con las métricas para interpretar el modelo. (Erroz, 2019)

2.2.14.1. **Matriz de confusión**

Generalmente la matriz de confusión es muy utilizada en el para evaluar modelos de toda la rama del machine learning, ya que es una tabla que muestra los verdaderos

positivos, verdaderos negativos, falsos positivos y falsos negativos cruzándose entre las clases reales y las clases que se predijeron con el modelo. El nombre de Matriz de confusión proviene porque muestra si el sistema está confundiendo dos clases (Visa et al., 2011).

		CLASES PREDICHAS	
		clase = si	clase = no
CLASES REALES	clase = si	TP = True Positive	TN=True Negative
	clase = no	FP = False Positive	FN=False Negative

Dónde:

- TP: Número correcto de predicciones que la instancia es positiva.
- FN: Número incorrecto de predicciones que la instancia es negativa.
- FP: Número incorrecto de predicciones que la instancia es positiva.
- TN: Número correcto de predicciones que la instancia es negativa

Las métricas de evaluación existente en un modelo deep learning vienen dadas por la sensibilidad, especificidad, y precisión, para ello debemos definir los criterios de prueba y una puntuación de rendimiento adecuada.

Para hacer esto, necesitaremos tres cosas:

- Una comprensión de lo que nuestro modelo va a generar
- Una medida que podemos utilizar para cuantificar el rendimiento del modelo.

- Algunos datos objetivos que podemos usar para calificar el desempeño del modelo de acuerdo con nuestra medida

2.2.14.2. **Precisión (accurancy)**

Precisión es lo que se conoce en machine learning como “accurancy”, y este indicador será bueno cuando la dispersión de los datos es mínima, la precisión viene dada por la dispersión de los valores que se obtienen a través de repetidas mediciones, es decir mientras que las predicciones sean en su mayoría correctas a los datos reales entonces se acercara a la igualdad de 1 o 100% y por ende la precisión será alta.

La fórmula para ver la precisión es la siguiente:

$$precision = \frac{numero - correcto - de - predicciones}{numero - total - de - imagenes}$$

2.2.14.3. **Exactitud**

En ciencia de datos está relacionada con el sesgo de la estimación, por lo tanto, se refiere a cuan cerca esta del resultado de medición del valor verdadero. La fórmula está determinada por los verdaderos positivos que fueron estimados por el modelo sobre la cantidad total de verdaderos positivos que viene a ser los verdaderos positivos más los falsos positivos.

$$Exactitud = \frac{TP}{TP + FP}$$

2.2.14.4. **Sensibilidad/true positive rate/recall (tpr)**

La sensibilidad del modelo es muy utilizada para estimar la tasa de verdadero positivos en ingles llamado como “recall” y es la proporción de todos los casos positivos, que fueron correctamente identificados en el modelo, este parámetro nos indica como de bueno es el modelo encontrado todo los positivos. La fórmula viene determinada de la siguiente manera:

$$TPR = \frac{TP}{FN + TP}$$

2.2.14.5. **False positive rate (fpr)**

La tasa de Falsos Positivos, en inglés “False Positive Rate” es la proporción de casos negativos que fueron erróneamente clasificados como positivos por el modelo. Se calcula según la siguiente ecuación.

$$FPR = \frac{FP}{FP + TN}$$

2.2.14.6. **F1-score/puntuación f1**

La puntuación F1 “F1- Score” es el que va a indicarnos la precisión del modelo en general para ello se toma en cuenta los resultados obtenidos en la precisión “Accunracy” y también la sensibilidad del modelo “Recall”, su cálculo viene determinado de la siguiente manera:

$$F1 = \frac{2 * precicion * recall}{precision + recall}$$

2.2.14.7. **Tasa de error**

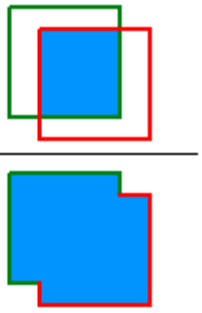
La tasa de error muestra que proporción de la data identifica incorrectamente. Su ecuación es la siguiente:

$$MissClasificationRate = \frac{FP + FN}{TP + TN + FP + FN}$$

2.2.14.8. **Intersección sobre unión**

En este indicador se utiliza básicamente para detección de objetos donde el cálculo se realiza del área de intersección del objeto sobre la unión total y este resultado se compara mediante un umbral que se define ya que el resultado saldrá entre 0 y 1 y el umbral puedeser 0.5 o 0.75 por ejemplo y dependiendo del resultado se definirá si es TP, FP, FN.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



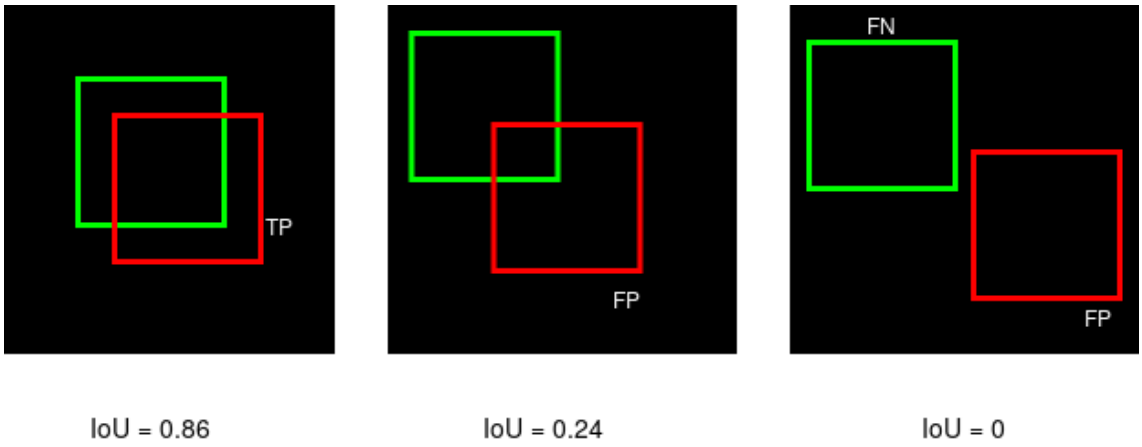


Figura 11. Intersección sobre Unión de cajas bajo un umbral x.

2.2.14.9. **Precisión media**

Simbolizado como AP@ que significa la precisión promedio del objeto identificado y esto esté sujeto al área bajo la curva IoU con un umbral determinado.

$$AP@\alpha = \int_0^1 p(r)dr$$

2.2.14.10. **Precisión media map@**

Observación (AP y número de clases): El AP se calcula individualmente para cada clase. Esto significa que hay tantos valores AP como número de clases (libremente). Estos valores de AP se promedian para obtener la métrica: Precisión promedio media (mAP). Precisamente, la precisión media (mAP) es el promedio de los valores de AP en todas las clases.

$$mAP@\alpha = \frac{1}{n}\sum_{i=1}^{n} AP_i \ \ for \ \ n \ \ \ classes$$

2.2.15.**Transfer learning**

Para entrenar un modelo desde cero primeramente se requiere bastante cantidad de imágenes para que pueda aprender a detectar bordes, sombras, contornos entre otras técnicas básicas de imágenes y por ello los pesos de las capas empiezan a asignarse aleatoriamente y además para empezar a aprender las funciones básicas se requerirá bastante procesamiento de GPU y RAM, al no contar con lo primero ni lo segundo se aplica una técnica llamada trasferencia de aprendizaje, esto implica que ya se tiene un modelo de ajuste previamente entrenado en un gran conjunto de datos y los pesos están configurados debidamente en cada capa, esta técnica hace que los pesos ya no empiecen aleatoriamente sinos ya sabiendo detectar contornos, o bordes de la imagen y por ello el aprendizaje se hace más rápido.

Es un enfoque popular en el aprendizaje profundo en el que los modelos previamente entrenados se utilizan como punto de partida en la visión por computadora y las tareas de procesamiento del lenguaje natural, dados los vastos recursos informáticos y de tiempo necesarios para desarrollar modelos de redes neuronales sobre estos problemas y a partir de los enormes saltos en las habilidades (Ruder, 2017).

2.2.16.**Arquitectura de redes clásicas (2 fases)**

2.2.16.1. **LENET-5**

La arquitectura Lenet-5 es la más antigua de todas y es la que propuso Lecun en el año de 1998 y la que en un inicio era solo dado como teoría ya que no se contaba con la suficiente capacidad de procesamiento para aplicarlo, por ello es la primera arquitectura que se aplicó en las redes neuronales convolucionales (Lecun et al., 1998).

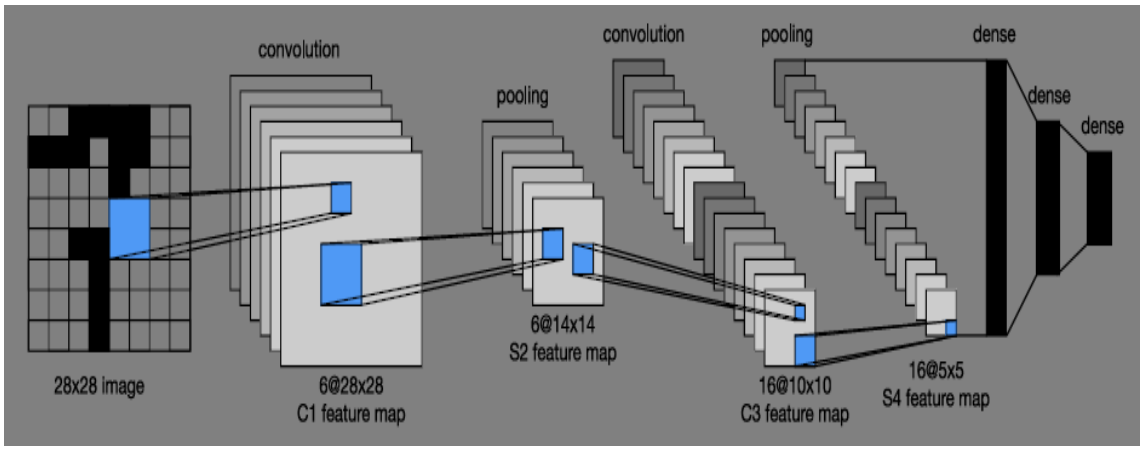


Figura 12. Arquitectura LeNet-5
Fuente: (Lecun et al., 1998)

2.2.16.2. **ARQUITECTURA ALEXNET**

Es otra de las arquitecturas antiguas, se remonta la aparición al año 2012, está compuesto por 5 capas convolucionales y 3 capas completamente conectadas por lo tanto viene a ser la menos profunda, se desarrolló así para dar un rápido procesamiento de imágenes por el mismo hecho de no contar con suficiente recursos computacionales, pero aun así fue la más efectiva para ese entonces ya que gano la competencia ILSVRC del 2012 que consistía en clasificar imágenes en la base de datos Imagenet significativamente, marcando algo importante en el desarrollo de arquitecturas de aprendizaje profundo. A pesar de ser la arquitectura más antigua de las estudiadas sigue siendo empleada por su gran funcionamiento (Krizhevsky et al., 2007)

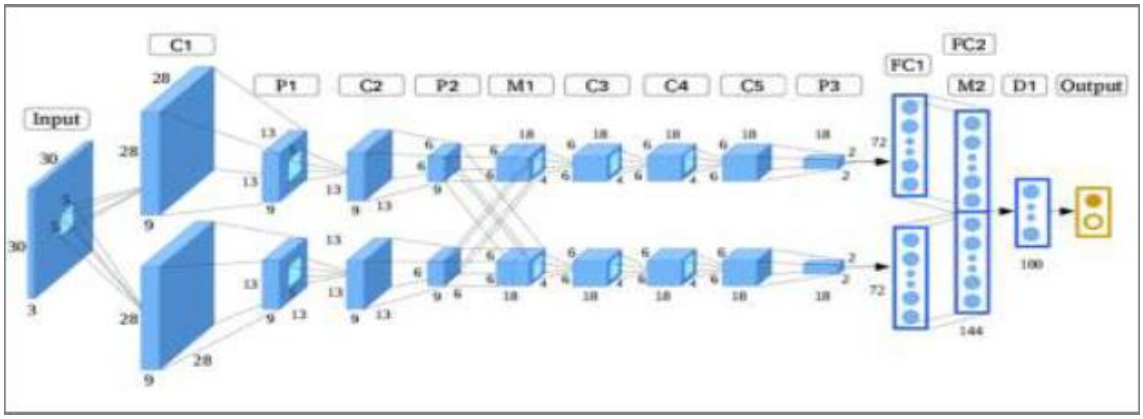


Figura 13. Ilustración de la Arquitectura AlexNet
Fuente: (Krizhevsky et al., 2007)

2.2.16.3. **VGG-16**

La arquitectura VGG-16 ha sido ampliamente utilizada en visión por computador en los últimos años donde se compone una apilada capa convolucionales y de agrupación máxima es utilizada por ser una arquitectura de 16 capas más pequeña y por lo tanto más rápida y conocida como VGG-16(Simonyan & Zisserman, 2015).

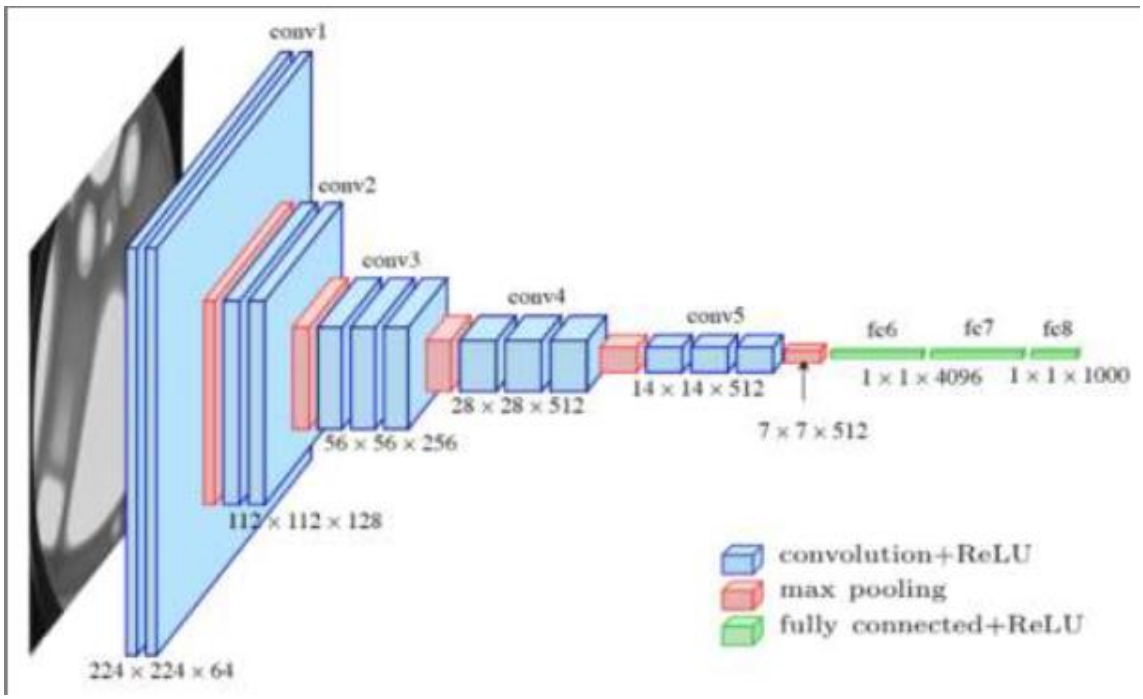


Figura 14. Arquitectura VGG-16
Fuente:(Simonyan & Zisserman, 2015)

2.2.17.**Arquitectura de redes modernas (1 fase)**

2.2.17.1. **Inception**

Esta arquitectura fue la que ocupo en primer lugar en la competencia ImageNet del año 2014, donde el objetivo fue detectar y clasificar e imágenes de esa gran base de datos, fue desarrollada por el equipo de Google en el mismo año de la competencia.

Esta arquitectura de aprendizaje profundo es un poco compleja, empieza a un inicio con cantidad de convoluciones a diferentes escalas y luego estos resultados son aumentados.

Los filtros que se utilizan son de 1x1, 3x3, 5x5; estos filtros extraen características a diferentes escalas, seguidamente se utiliza la agrupación máxima; para que la salida quede correctamente con las mismas dimensiones (Szegedy et al., 2014)

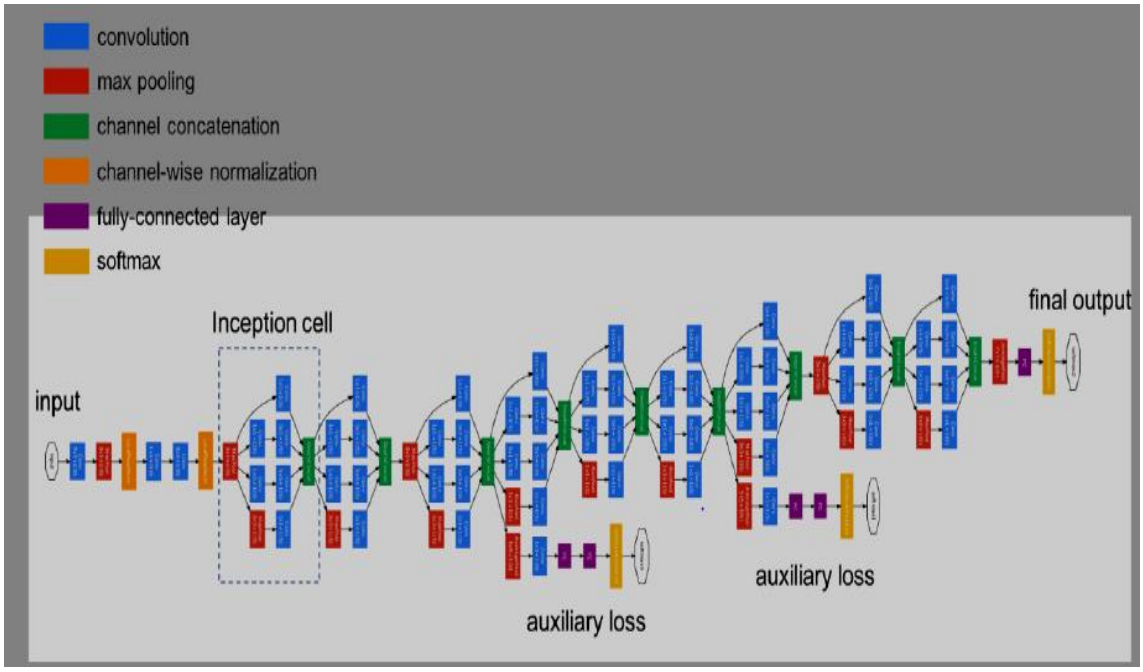


Figura 15. Arquitectura general de Inception
Fuente: (Szegedy et al., 2014)

2.2.17.2. **RESNET**

Sabiendo que la cantidad de capas ocultas podría ser demasiado en busca de supuestamente incrementar la precisión del modelo, en la realidad no es así, ya que al existir gran cantidad de capas ocultas y utilizando optimizadores basados en gradientes se realiza una propagación hacia atrás, en ese proceso ocurre lo que se conoce como gradientes de desaparición por lo que los pesos dejan de actualizarse después de un determinado tiempo, y por consiguiente los gradientes explosivos son el fenómeno en el que se realizan grandes actualizaciones.

Por ello aparece la arquitectura Resnet, que ya no existe la preocupación de incrementar capas ocultas y ya no se da el problema de gradientes de fuga de explosión. Los bloques

residuales al realizar la propagación hacia atrás se mantienen almacenado para que si en el siguiente bloque no se actualiza o aprende nada se pueda reutilizarlo almacenado.

Por tanto, los bloques residuales pueden usarse junto con las capas convolucionales para maximizar la precisión. Antes de realizar un cambio en dichos pesos estos se copian y se crean como accesos directos, y esto se pueda reutilizar en el caso de que al realizar alguna operación de convolución y se obtenga un resultado no esperado y sea lo que cause el cambio de todo el modelo, es ahí donde se reutilice el peso anterior y/o también se puede agregar y realizar una combinación entre ambos pesos y se elimine por completo los efectos negativos que pudiera tener, entonces mediante la eliminación de estos efectos negativos que se puedan tener en el modelo, son esos pesos copiados y guardados como una función de identidad (He et al., 2015)

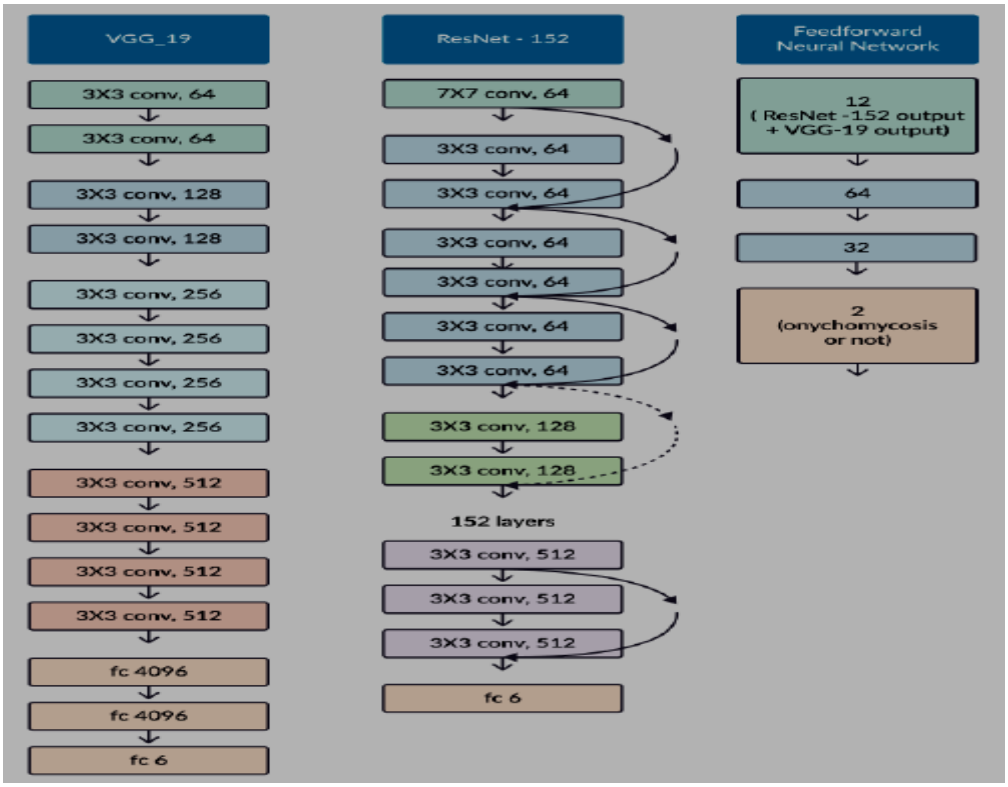


Figura 16. Arquitectura de ResNet
Fuente: (He et al., 2015)

2.2.17.3. **DENSENET**

Como indica el nombre de esta red, es una red densa, y en este caso se arquitectura se utiliza cuando se tiene un mapa de características desde antes de la red, esto quiere decir que el mapa de características se utiliza con la concatenación a la entrada de cada capa posterior formando así lo que se denomina un bloque denso, esto hace que se pueda sacar beneficio directo de las características de capas anteriores. Los autores afirman que "la concatenación de mapas de características aprendidos por diferentes capas aumenta la variación en la entrada de las capas posteriores y mejora la eficiencia"(Huang et al., 2018)

De acuerdo a los autores afirman que DenseNet logra un mejor rendimiento con menos complejidad que ResNet.

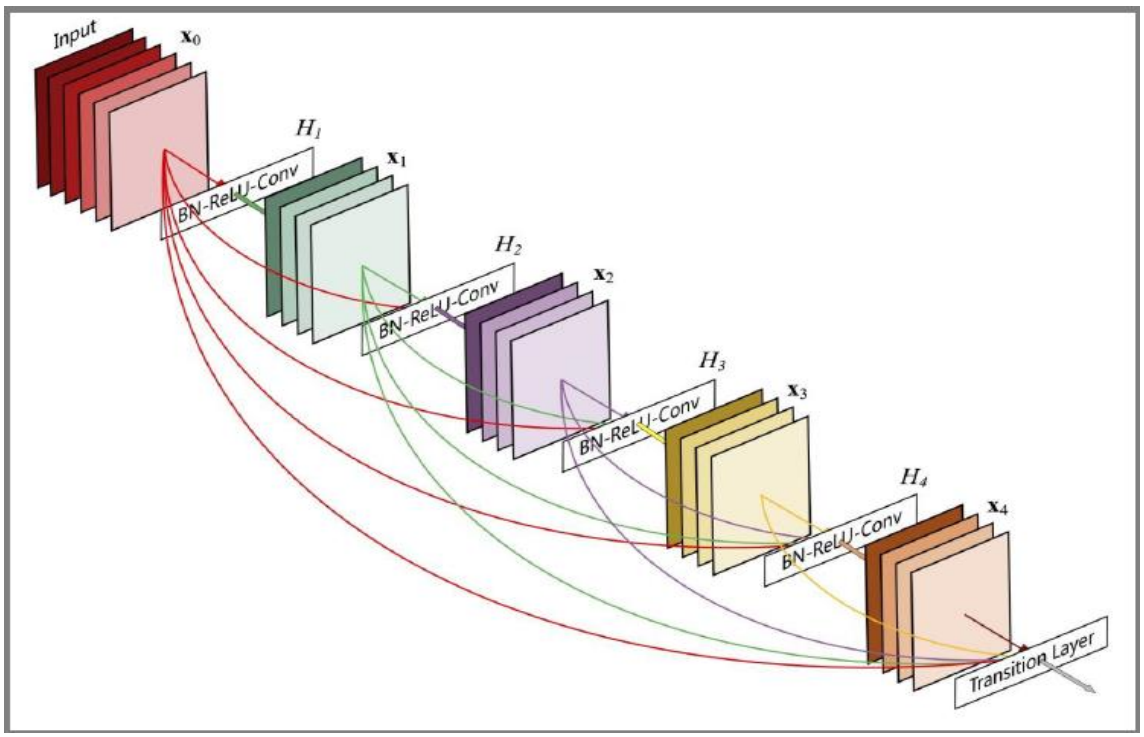


Figura 17. Arquitectura de DenseNet
Fuente: (Huang et al., 2018).

2.2.17.4. **YOLO**

El problema de las anteriores arquitecturas mencionadas es que el entrenamiento tarda demasiado tiempo en la inferencia y esto ocasionaba que no podía realizarse detecciones en video o si se lograba era con un tiempo demasiado largo que realizaba, es por ello que desarrollaron una arquitectura que la principal característica que tiene es que todo se realiza en una sola etapa, este algoritmo se basa en la regresión y el mismo instante que se va ejecutando la regresión también va identificando las coordenadas del cuadro que se delimita y asignando probabilidades de identificación de acuerdo a una determinada clase dada. (Redmon et al., 2016)

La primera versión de la arquitectura YOLO llego a procesar imágenes en tiempo real a 45 fps y luego realizando una variante de esto modelo que fue denominado FAST Yolo llegó a procesar 155 fps (Redmon et al., 2016).

2.2.17.5. **YOLOV4**

Después de la implementación de Yolo V1 por el año 2015, aparece las siguientes versiones de YOLOV2, YOLOV3 y YOLOV4 en el mes de abril 2020, y actualmente viene desarrollándose la versión de YOLOV5, es por ello que al ser uno de las arquitecturas que supera ampliamente en la detección de objetos, se optó por trabajar con esta para la presente investigación.

En esta versión implementaron nuevos módulos, aplicaron la técnica de data augmentation y al combinar todos ellos obtuvieron un mAP de 65.7% en el gran conjunto de datos de MSCOCO y en la detección de videos tuvieron resultados a una velocidad de 65 fps en tesla V100(Bochkovskiy et al., 2020).

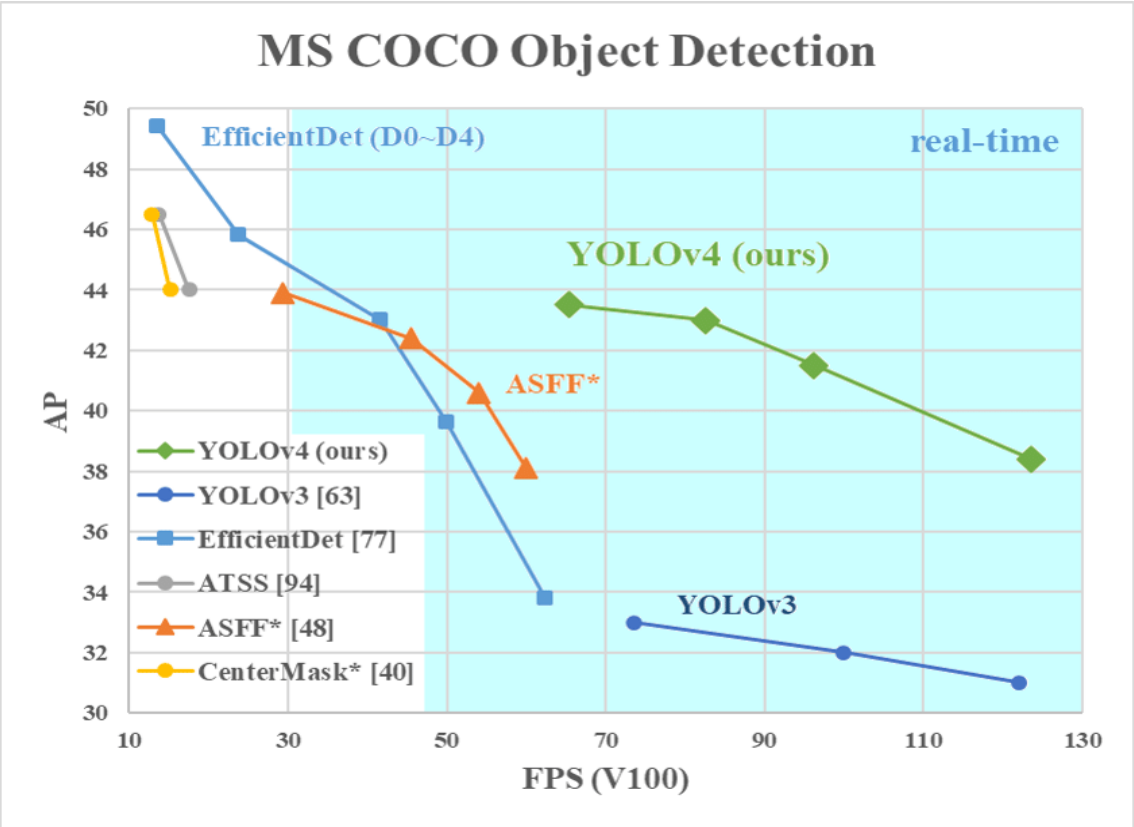


Figura 18. Comparación de mAP entre diferentes arquitecturas

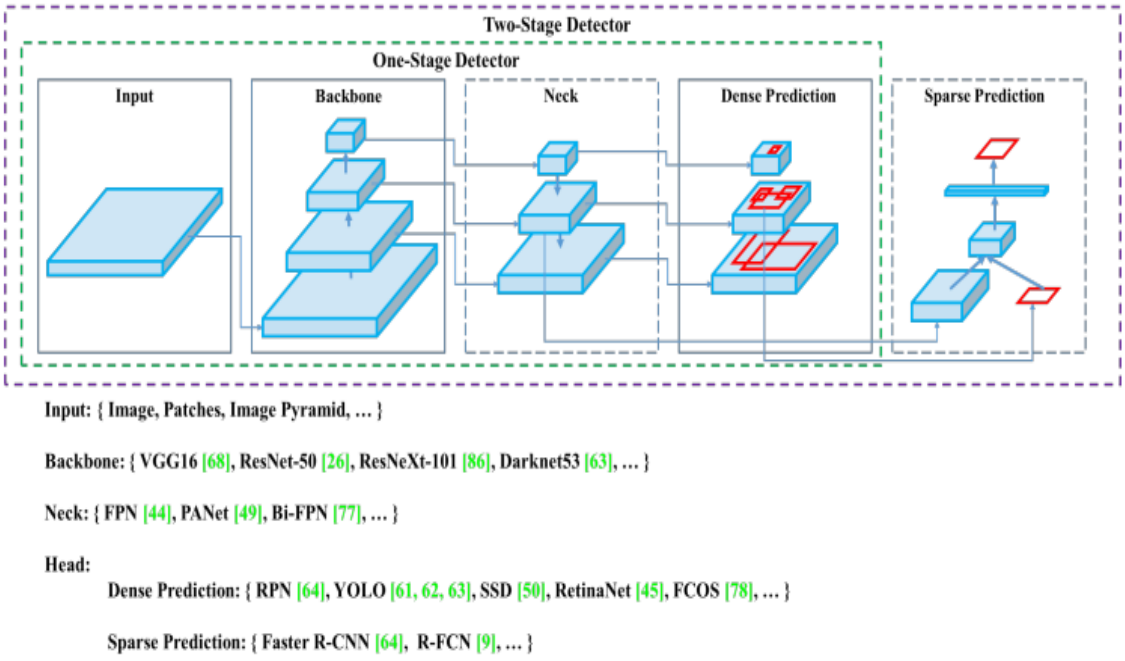


Figura 19. Arquitectura de YoloV4
Fuente: (Bochkovskiy et al., 2020)

YOLOv4 se utiliza en varios campos y tareas:

Gobierno de Taiwán: control de tráfico www.taiwannews.com.tw/en/news/3957400 y youtu.be/IiU6wFmfVnk

Amazon: Asistente a distancia Anti-Covid19 github.com/amzn/distance-assistant e instancias Inf1 de Amazon Neurochip / Amazon EC2: aws.amazon.com/ru/blogs/machine-learning/improving-performance-for-deep-learning-detección-de-objetos-basada-con-una-neurona-aws-compilada-yolov4-modelo-en-aws-inferentia

Laboratorio de innovación de BMW: github.com/BMW-InnovationLab

Y en muchas otras tareas….

2.2.17.6. **EFFICIENTDET**

Otro modelo de la familia de detección de un solo paso y fue desarrollado por un equipo de Google donde el propósito principal es tener una eficiencia mayor respecto a las demás arquitecturas, primeramente se enfoca en una red de pirámide de características bidireccionales, y luego se encamina en un método que escala resolución, profundidad y ancho para todas las redes troncales realizando al mismo tiempo el contorno de las cajas de detección y las probabilidades que estas tienen, realizando el entrenamiento y validación en el conjunto de datos COCO obtiene 55.1 AP con 77M parámetros y 410B FLOP1 , siendo 4 veces más pequeño y usando 13 veces - 42 veces menos FLOP que los detectores anteriores.(Tan et al., 2020)

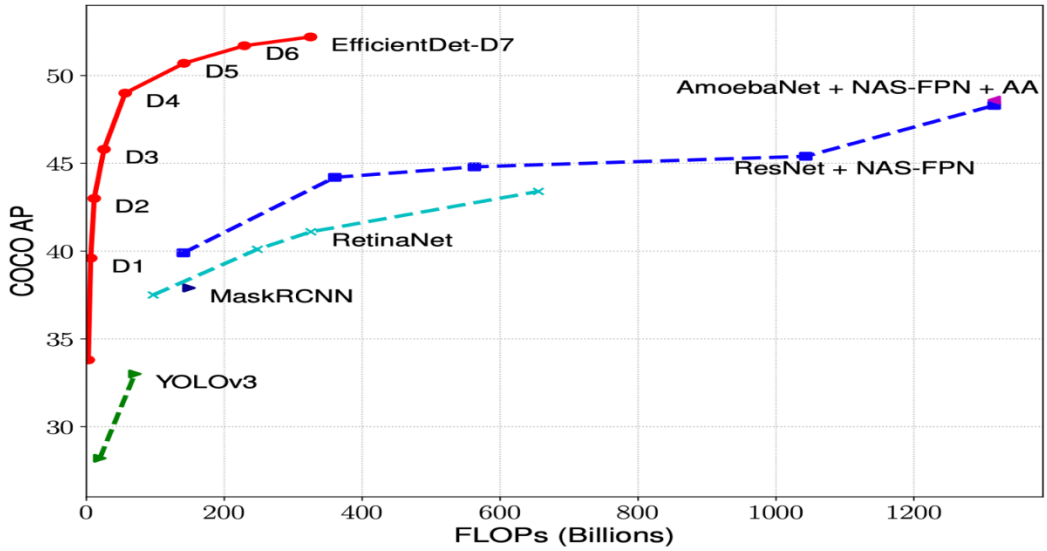


Figura 20. Comparación de efficient det con otras arquitecturas modernas

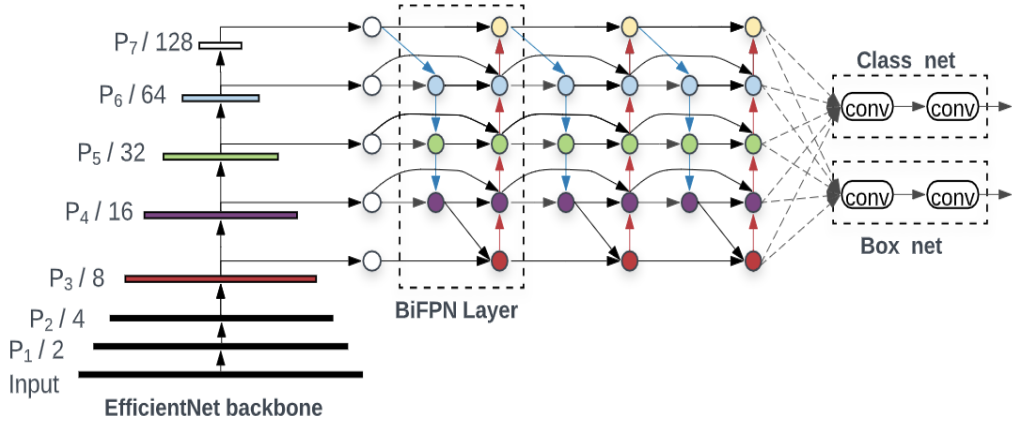


Figura 21. Arquitectura del modelo EfficientDet.
Fuente: (Tan et al., 2020)

2.2.17.7. **YOLO V5**

Glenn Jocher es considerado el autor de YoloV5, pero todo el código está en el repositorio de Ultralytics LLC. Debe reconocerse que los chicos estaban comprometidos con el soporte de YoloV3 y la portabilidad a iOS, así como a la versión 5. Tenga en cuenta que los repositorios Ultralytics YOLO V3 y V5 no son bifurcaciones del original, a diferencia del repositorio de Alexey.

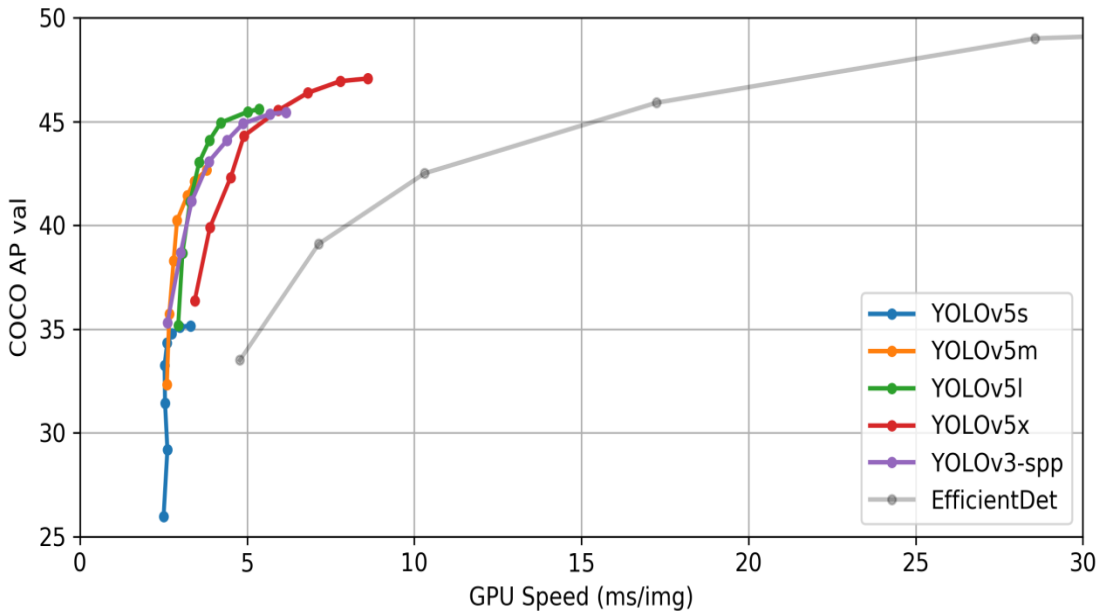


Figura 22. Comparación de YoloV5 con EfficientDet
Fuente :(Nelson & Solawetz, 2021)

Existe una controversia por esto de la implementación de YoloV5 y haciendo un poco de historia del origen de esta arquitectura, el autor Glenn creo el repositorio llamando YoloV5 en fecha 29 de mayo del 2020, luego el 09 de junio confirmo mediante un mensaje la implementación de dicho modelo.(Nelson & Solawetz, 2021).

La implementación de YoloV5 difiere de las versiones anteriores en algunas formas notables. Primero, porque aún no existe un artículo oficial de la mencionada arquitectura, en segundo lugar, implementó YoloV5 de forma nativa en PyTorch, mientras que todos los modelos anteriores de la familia YOLO estaban desarrollados en la arquitectura darknet (Nelson & Solawetz, 2021).

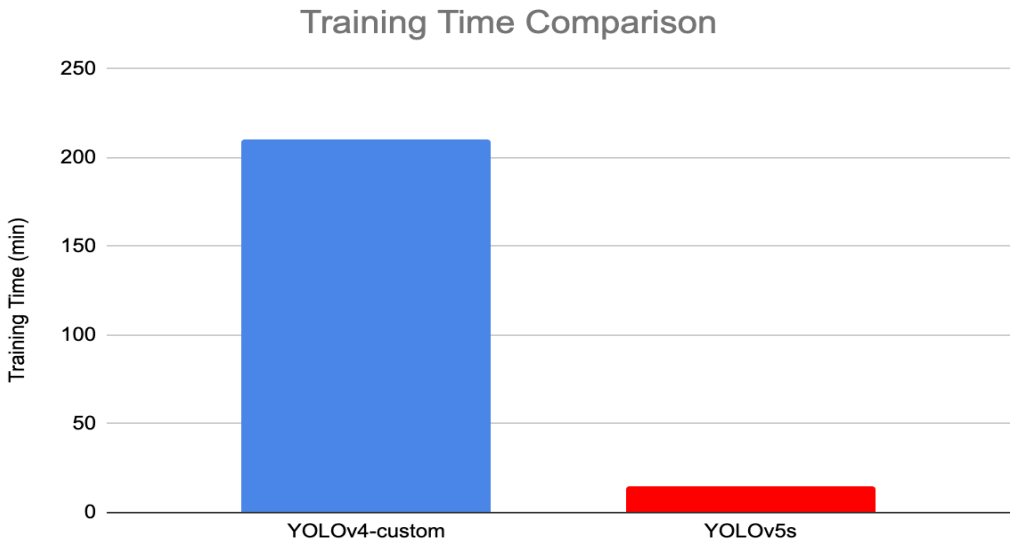


Figura 23. Tiempo de entrenamiento en YoloV5**.**

Tanto YoloV4 como YoloV5 implementan el cuello de botella CSP para formular características de imagen, con el crédito de investigación dirigido a WongKinYiu y su artículo reciente sobre Cross Stage Partial Networks para la red neuronal convolucional. El CSP aborda problemas de gradientes duplicados en otras redes troncales de ConvNet más grandes, lo que resulta en menos parámetros y menos FLOPS para una importancia comparable. Esto es extremadamente importante para la familia YOLO, donde la velocidad de inferencia y el tamaño pequeño del modelo son de suma importancia (Nelson & Solawetz, 2021).

Los modelos CSP se basan en DenseNet. DenseNet fue diseñado para conectar capas en redes neuronales convolucionales con las siguientes motivaciones: aliviar el problema del gradiente de desaparición, reforzar la propagación de características, alentar a la red a reutilizar características y reducir el número de parámetros de red.

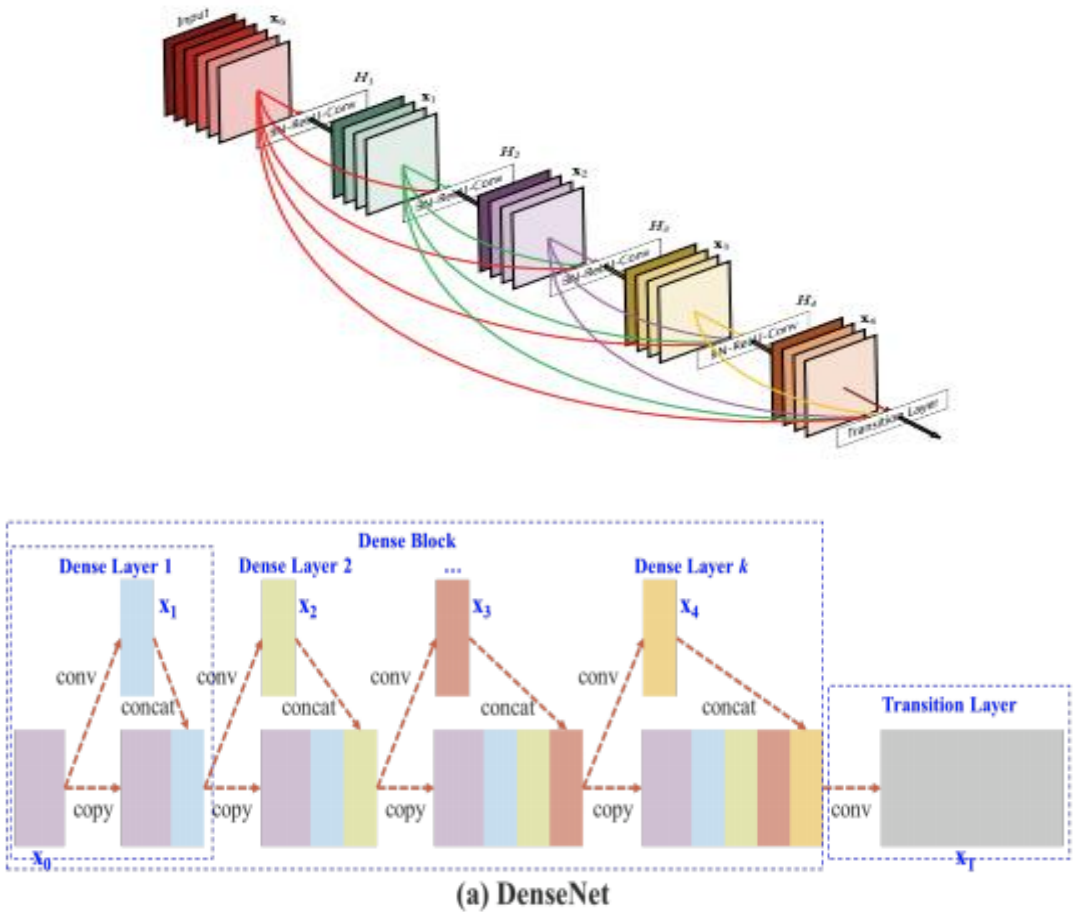


Figura 24. Arquitectura planteada en YoloV5.
Fuente: (Nelson & Solawetz, 2021)

YoloV4 y YoloV5 realizan el cuello PA-NET para la agregación de funciones.

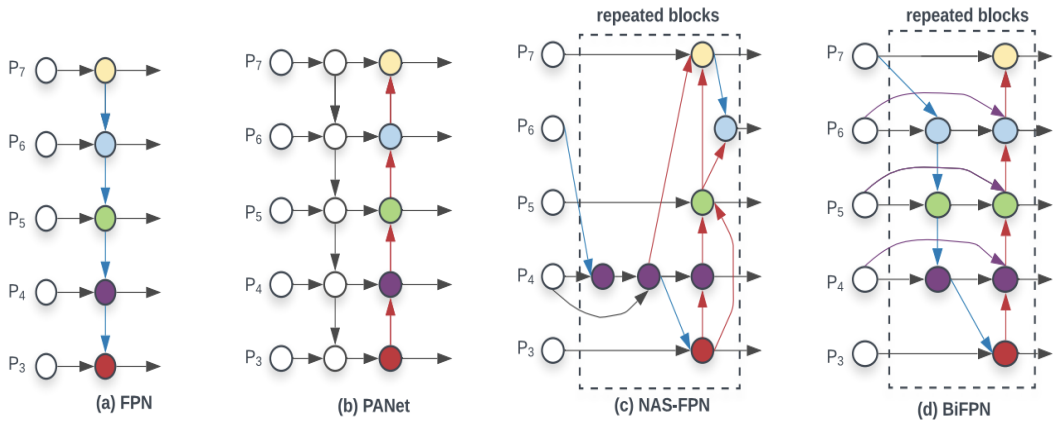


Figura 25. Capas de arquitectura YoloV5

2.2.18.**Lenguaje de programación**

El lenguaje de programación son códigos que se escriben manteniendo un orden y una lógica en su estructura, la característica más importante de los lenguajes de programación es que cualquier persona que sea programador pueda realizar un sistema o software específico y esta misma pueda ser comprendido por otro programador y ya lo pueda modificar, incrementar, reutilizar o apoyar en la construcción de dicho sistema (Por & Aproximación, 2019)

2.2.19.**Visión por computadora**

La visión por computadora se le conoce también como visión artificial, es una rama de la inteligencia artificial que cuenta con métodos y teorías formuladas simulando a la visión biológica humana. A través de diversas técnicas extrae y analiza información que obtiene de las imágenes y últimamente de los videos; también la visión por computadora permite manipular y tomar datos de diferentes tipos de imágenes como eliminación de objetos en una imagen, construcción de una imagen a partir de una parte, imágenes satelitales, reconocimiento de rostros, entre otros todo ello realizando automáticamente, la tarea de realizar estas a funciones es un poco complicado para una maquina programada sin visón computacional, pero aplicando todas las teorías y técnicas de la visión computacional con un análisis cuantitativo se le hace muy fácil realizar todas estas tareas. (Viera, 2017)

Los procesos de la visión artificial pueden agruparse en 3 niveles según su complicación e implementación:

- Procesos de bajo nivel: son operaciones primitivas que se realizan en una imagen como eliminar ruido, contraste, nitidez; la entrada es una imagen y la salida también es otra imagen.
- Procesos de nivel medio: aquí es donde ya se realiza una operación en específico como puede ser descripción de características en una imagen, segmentación, por ende, las entradas en este proceso son imágenes, pero las salidas son un conjunto de características extraídas como contornos, bordes, entre otros.
- Procesos de nivel superior: En este proceso se realizan mediante técnicas avanzadas de visión artificial mediante diferentes operaciones en una imagen simulando a la visión biológica como por ejemplo darle como entrada una imagen y el programa pueda interpretarlo de acuerdo a la imagen dada.

2.2.20.**Elementos De Visión Por Computadora**

2.2.20.1. **Sistema De Iluminación**

La iluminación es un aspecto muy importante al momento de realizar operación con visión artificial; ya que teniendo un buen enfoque de iluminación se puede lograr mejores precisiones en obtener resultados; la característica más importante del sistema de iluminación es la intensidad que tiene la luz y el sentido de iluminación que se mantenga en un solo lado, es por ello que grandes laboratorios o empresas que trabajan con visión computacional tiene especial tratamiento con la iluminación por ello diseñan un módulo específico de iluminación exclusivamente para que resalte las particularidades del producto. A pesar de que existen algoritmos para corregir fallas en la iluminación; tener un buen sistema de iluminación ayuda bastante en las operaciones

a realizarse y se ahorra en tiempo y precisión ya que puedan existir sombras; bajo contraste entre otros; por ello mantener un buen sistema de iluminación en una imagen es de mucha importancia y se resalta las características importantes que tenga la imagen (Viera, 2017).

Se clasifican en:

Dependiendo de la intensidad de la luz

Dependiendo de la dirección de la iluminación

Dependiendo de la fuente de iluminación

2.2.20.2. **Cámaras Y Tarjetas De Captura**

En esta investigación se usó la cámara de un celular de marca Huawei mate 20 lite:

2.2.20.3. **Pixel**

Cada imagen consiste en un conjunto de pixeles. Los pixeles son la unidad básica de análisis en una imagen. No hay un elemento de menor dimensión y más fino en una imagen que el pixel. Se suele relacionarlo con “color” o “intensidad”. Si pensamos en una imagen como una cuadrícula, cada cuadrado en la cuadrícula contiene un solo pixel. Por ejemplo, una imagen con una resolución de 500 x 300 significa que dicha imagen está conformada por una matriz de pixeles con 500 filas y 300 columnas. La mayoría de los pixeles se representan de dos maneras: escala de gdrises y color. En una imagen en escala de grises, cada pixel tiene un valor entre 0 y 255, donde cero corresponde al color negro y 255 al color blanco. Los valores entre 0 y 255 representan variaciones de los tonos de gris, donde los valores cercanos a 0 son más oscuros y los cercanos a 255 son más claros. Los pixeles de color se representan normalmente en espacios de color; el

más utilizado de ellos es el denominado RGB, compuesto por un valor para el componente rojo, uno para el verde y uno para el azul. Cada uno de estos tres colores está representado por un número entero en el rango 0 a 255. Dado que el valor de pixel solo necesita estar en el rango [0, 255] se usa normalmente un entero sin signo de 8 bits para representar cada intensidad de color. Como referencia, a continuación, se muestran en la tabla 2.1 ejemplos de colores y su vector RGB.

2.2.20.4. **Espacios De Color**

Entre los espacios de color más utilizados para el procesamiento de imágenes están el rojo, el verde y el azul (RGB). Según la Comisión Internacional de Iluminación, en su norma técnica CIE 1931-RGB, el color se conforma por tres dimensiones monocromáticas perceptibles al campo visual. Sin embargo, estos no son los únicos espacios de color; existen, entre otros, los espacios HSV (Hue-Saturation-Value) y los espacios L*a*b (es un espacio de color independiente del dispositivo que incluye todos los colores que pueden ser percibidos por los humanos.), más orientados a describir la percepción del color ante un cambio en alguna dimensión de la imagen. La elección de un espacio de color está en función del uso que se desea para la imagen, en caso de que fuera necesario hacer un análisis en relación con la superficie donde se utilizará. En este tipo de aplicaciones se emplea la combinación CMyK (Cyan, Magenta y Key) que tiene mucha utilidad para la comparación de patrones de impresión. Para el campo de procesamiento y análisis de pixeles en una imagen se utiliza en especial el espacio de color RGB. En la figura 2.1 se muestran diversas aplicaciones de espacio de color para una misma imagen (LAMEGO, 2017).

Código de colores para el espacio de color utilizados en OpenCV son las siguientes:

- CV_RGB2GRAY. Conversión de RGB a escala de Grises
- CV_BGR2HSV. Conversión de BGR a escala HSV
- CV_RGB2HSV Conversión de escala RGB a escala HSV
- CV_HSV2BGR Conversión de HSV a escala BGR
- CV_HSV2RGB Conversión de HSV a escala RGB
- CV_BGR2HLS Conversión de BGR a escala HLS
- CV_RGB2HLS Conversión de RGB a escala HLS
- CV_HLS2BGR Conversión de HLS a escala BGR
- CV_HLS2RGB Conversión de HLS a escala RGB

los cuales estarán en el algoritmo, cumplen la función de conversión de colores.

2.2.20.5. **Bordes**

La segmentación de una imagen es el proceso de dividir la imagen en piezas tal que diferentes objetos o partes de objetos son separados.

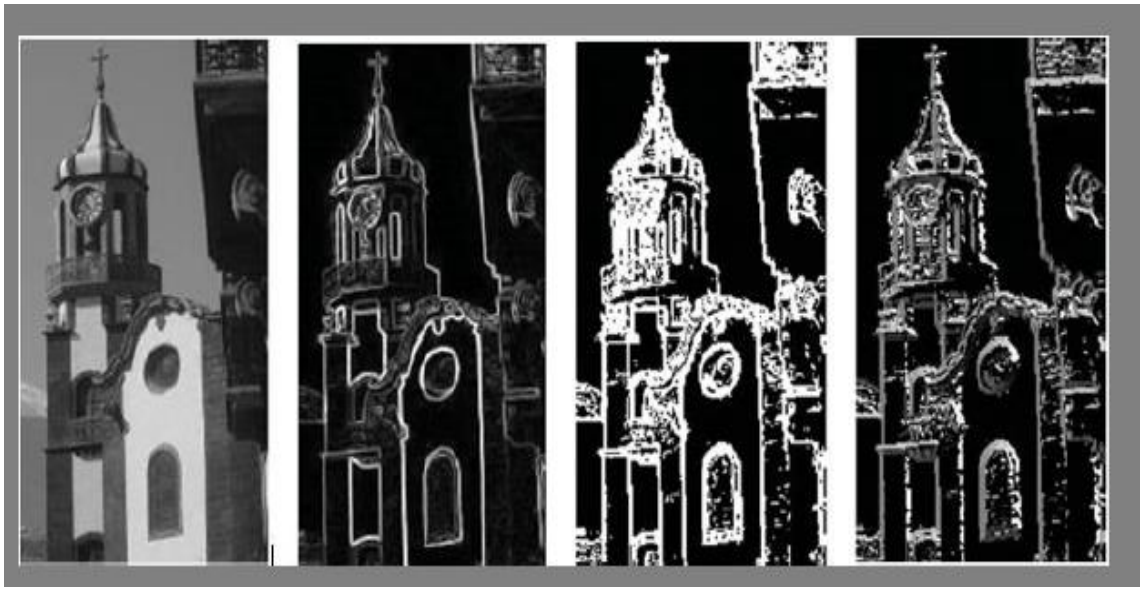


Figura 26. Detección de bordes

2.2.21.**Algoritmo**

Los algoritmos son una serie de normas o leyes específicas que hace posible la ejecución de actividades, cumpliendo una serie de pasos continuos que no le originen dudas a la persona que realice dicha actividad, un algoritmo es un conjunto de instrucciones paso a paso que manipulan la información para encontrar la solución a un problema. De hecho, algoritmo no es específico para las computadoras y tiene sus raíces derivadas de matemáticas. Casi todas nuestras actividades cotidianas que van desde multiplicar números a la programación de vuelos se basan en un conjunto definido de reglas se realizan de manera predefinida que constituyen el proceso algorítmico. (Olarte Gervacio, 2018).

2.2.22.**Google Colab**

Colab es un servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google, con librerías como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV. Todo ello bajo Python 2.7 y 3.6, que aún no está disponible para R y Scala.

Aunque tiene algunas limitaciones, que pueden consultarse en su página de FAQ, es una herramienta ideal, no solo para practicar y mejorar nuestros conocimientos en técnicas y herramientas de Data Science, sino también para el para el desarrollo de aplicaciones (pilotos) de machine learning y deep learning, sin tener que invertir en recursos hardware o del Cloud.

Con Colab se pueden crear notebooks o importar los que ya tengamos creados, además de compartirlos y exportarlos cuando queramos. Esta fluidez a la hora de manejar la información también es aplicable a las fuentes de datos que usemos en nuestros

proyectos (notebooks), de modo que podremos trabajar con información contenida en nuestro propio Google Drive, unidad de almacenamiento local, github e incluso en otros sistemas de almacenamiento cloud, como el S3 de Amazon.

Google Colab es un servicio en la nube totalmente gratuito que proporciona una GPU de hasta 12gb de memoria lo cual nos facilita entrenar modelos de deep learning a una gran velocidad, lo único necesario será tener una cuenta Gmail para sincronizar los archivos mediante el drive que te asigna hasta 15gb de almacenamiento de manera gratis; la única dificultad que se puede encontrar en este entorno de desarrollo es que cada 12h se reinicia automáticamente por ende se recomienda trabajar todos los modelos en la nube para ir almacenando la avanzando.

2.2.23.**Anaconda**

En de libre distribución y multiplataforma, fue creado específicamente para el data sciense, y el aprendizaje automático, es un entorno de desarrollo que se puede instalar fácilmente los paquetes necesarios para un proyecto y separar de otro proyecto creando otro entorno especifico, muy fácil y adaptable para utilizarlo.

2.2.24.**Python**

Python fue creado por Guido Van Rossum, hace más de 10 años, asimismo cabe indicar que él también fue el propulsor de otro lenguaje creado como ABC, este no era orientado a objetos por ello que creo Python ya que tiene soporte para crear programas orientados a objetos (Yegulalp, 2021)

Python es un lenguaje interpretado lo que quiere decir que se necesita compilar el código para poder ejecutarlo, es un lenguaje multiplataforma ya que se puede utilizar

desde aplicaciones Windows, servidores de red, páginas web, la que le hace diferente a los demás lenguajes de programación es que es fácil de aprenderlo, se puede escribir código de alguna instrucción en menos líneas lo cual la hace rápido y asimismo al momento de ejecutarlo se realiza con una velocidad mejor que los demás lenguajes (Yegulalp, 2021).

Python es un lenguaje de programación de alto nivel, por ende, existe la facilidad de escribir código en menos líneas en comparación a programas como C, C++o el mismo java, es muy recomendable para aquella persona que se está iniciando en el mundo de la programación.

2.2.25.**Librerías utilizadas de Python**

2.2.25.1. **Opencv**

OpenCV es una librería de código abierto dedicada exclusivamente para manipulación de imágenes que está desarrollada en la plataforma de C y C++ y es multiplataforma; para poder utilizarlo solo hay que descargarlo de la página http://opencv.org. Existen interfaces disponibles para diferentes lenguajes de programación como Python, java, Ruby y otros. Existen más de 500 funciones en esta librería, generalmente todas ellas trabajan en tiempo real, optimizando los recursos en cada nivel, desde los algoritmos hasta el uso de múltiples núcleos en paralelo, utiliza un módulo GPU facilitada por CUDA también utiliza OpenCL (GPU genérica). El principal objetivo de OpenCV es que cualquier persona principiante en la programación pueda utilizarlo fácilmente y se pueda familiarizar de manera más rápida para que pueda crear cualquier aplicación de visión por computadora. Dentro de las funciones de OpenCV se encentran como por ejemplo manipulación de imágenes médicas, seguridad, inspeccionar productos de

fábrica, robótica entre otros muchos que la hacen una potente librería; asimismo tiene una sub-librería especial para aprendizaje automático denominado “MLL”, junto con esta sub-librería se puede desarrollar aprendizaje automático en imágenes, y es suficientemente bueno para cualquier modelo de aprendizaje automático en conjuntos de imágenes (Kaehler & Bradski, 2016).

Transformaciones morfológicas en opencv

Las transformaciones que se puede realizar con OpenCV son operaciones que tiene esta librería y se considera como básicas en la manipulación de imágenes; para ello es necesario que la imagen este convertida en imágenes binarias. Requiere de dos entradas; una que es la propia imagen y otra que se denomina como kernel o núcleo que será el encargado de transformar la imagen. Existen varios kernel para trasformar imágenes entre los más básico y utilizaos son la dilatación y la erosión (Dawson-Howe, 2014).

Dilatación

La dilatación es una técnica para expandir el número de píxeles del objeto,

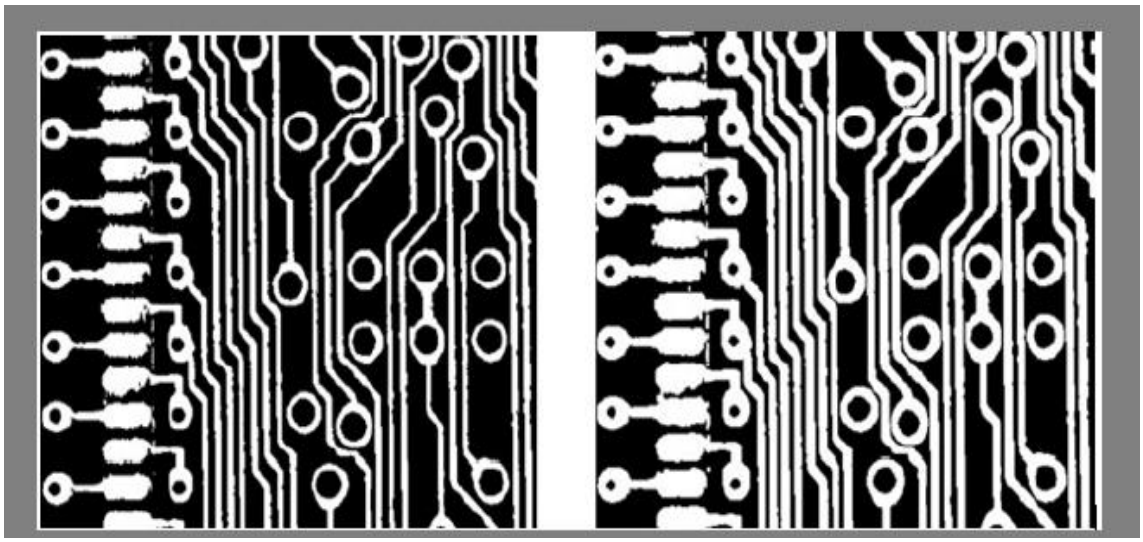


Figura 27. Imagen binarizado y luego dilatado
Fuente: (Dawson-Howe, 2014)

Erosión

La erosión es una técnica para reducir las formas de los objetos mediante la eliminación de píxeles.

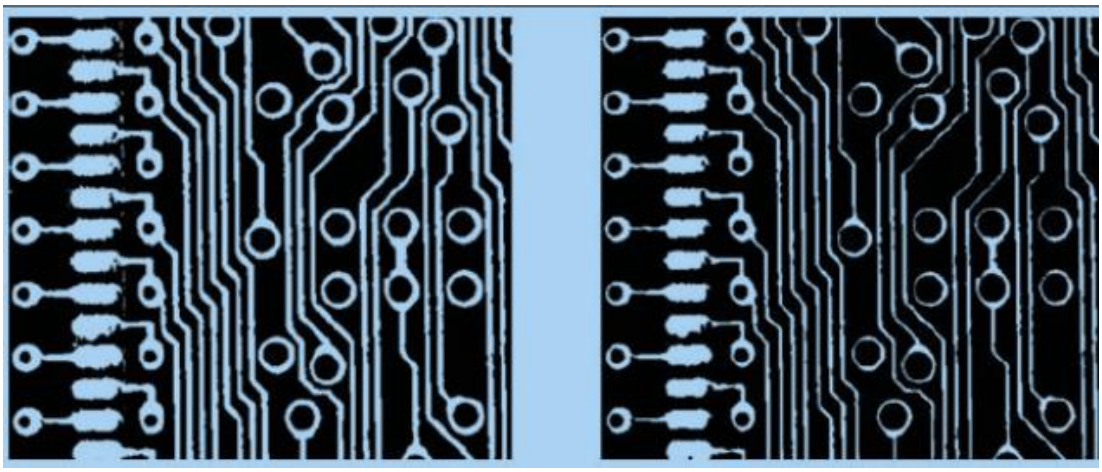


Figura 28. Imagen binarizado y luego erosionado
Fuente: (Dawson-Howe, 2014)

2.2.25.2. **Numpy**

NumPy es una biblioteca Python que proporciona tipos de datos para almacenar de manera eficiente secuencias de valores numéricos y operar sobre ellos. NumPy almacena estos valores numéricos con un tamaño fijo y los almacena en regiones contiguas de memoria, lo que permite una comunicación sencilla con otros lenguajes de programación como C, C++ y Fortran (de hecho, varias funciones de la biblioteca están escritas en estos lenguajes para maximizar el rendimiento). Sobre estos datos, NumPy permite realizar operaciones matemáticas del álgebra lineal o algoritmos más avanzados como la transformada de Fourier. Desde el punto de vista de tipos de datos, NumPy ofrece ndarray, un array ndimensional de elementos el mismo tipo. El acceso a estos elementos se realiza de manera natural mediante sus coordenadas en el espacio n-dimensional. NumPy ofrece distintos tipos de datos numéricos (Caballero Roldán, Martín Martín, & Riesco Rodríguez, 2018).

2.2.25.3. **Matplotlib**

Matplotlib es una biblioteca de trazado 2D de Python que produce cifras de calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas. Matplotlib se puede usar en scripts de Python, los shells de Python e IPython, el cuaderno Jupyter, los servidores de aplicaciones web y cuatro kits de herramientas de interfaz gráfica de usuario. Matplotlib intenta hacer que las cosas fáciles sean fáciles y las cosas difíciles posibles. Puede generar gráficos, histogramas, espectros de potencia, gráficos de barras, gráficos de errores, gráficos de dispersión, etc., con solo unas pocas líneas de código. (Hunter, Dale, Firing, & Droettboom, 2019)

2.2.26.**Proyecto De Instalación De Banda Ancha**

El proyecto denominado instalación de banda ancha para la conectividad integral y desarrollo social de las regiones Junín, Puno, Moquegua y Tacna fue adjudicada por pro inversión al consorcio denominado OROCOM S.A.C. en el año 2017 y que este consorcio está conformado por Eléctricas de Medellín Perú, Amitel Perú Telecomunicaciones SAC, Tuenza internacional corporation.

Las regiones demandaran una extensión aproximada de 5389 km de fibra óptica con llegada a 623 mil habitantes beneficiando a 1097 locales escolares de gestión estatal 559 establecimientos de salud y 70 dependencias policiales

En la región de Puno se tiene a 471 localidades beneficiarias que consisten en 285 establecimientos de salud 38 dependencias policiales y 635 locales escolares, beneficiando a un total de la población de 287,822 habitantes con una extensión aproximada de fibra óptica de 2556 km. (PROINVERSION, 2018)

Figura 29. Entidades ejecutoras del proyecto regional de banda ancha

Figura 30. Beneficiarios del proyecto en las localidades.

2.2.26.1. **Infraestructura**

- Red de transporte: es la red mediante la cual se trasladará la fibra óptica (cable de datos resistente,). Esta red es trasladada desde la capital de la provincia a la capital distrital mediante postes de media tensión, baja

tensión y en tramos que no existe posteria se plantan postes nuevos, es en esta etapa de implementación donde se ejecuta el presente trabajo de tesis.

- Red de acceso: Es la red que permite brindar la señal de internet. Puede ser alámbrica o inalámbrica. Parte desde la capital del distrito a la localidad.
- Una vez que llega a la localidad, los principales beneficiarios serán las instituciones. Sin embargo, el servicio ya está en la comunidad lo que significa que una vez instalado podrán requerir el servicio.

2.2.27.**Herrajes Instalados En Postes**

2.2.27.1. **Cruceta**

Cruceta telefónica, para el almacenamiento de cable ADSS sobrante. Ideal para la instalación en poste o torre. Diseño y sujeción al poste que permite reducir tensión del cable al momento de almacenamiento

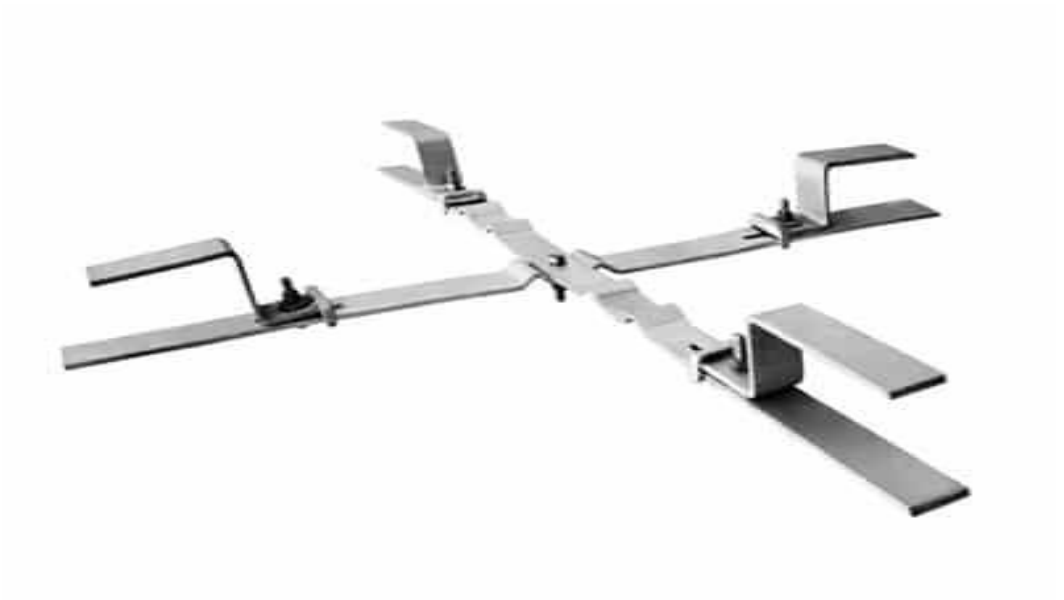


Figura 31. Cruceta

2.2.27.2. **Preforme De Retenida Vano 200**

Es un preforme de anclaje para vanos menores a 200m de distancia, vienen medidas del preforme de diferente distancia y para cada tipo de ADSS dependiendo de la mili metraje, pero en todos ellos siempre tienen la misma apariencia y va instalado junto a un guardacabo.

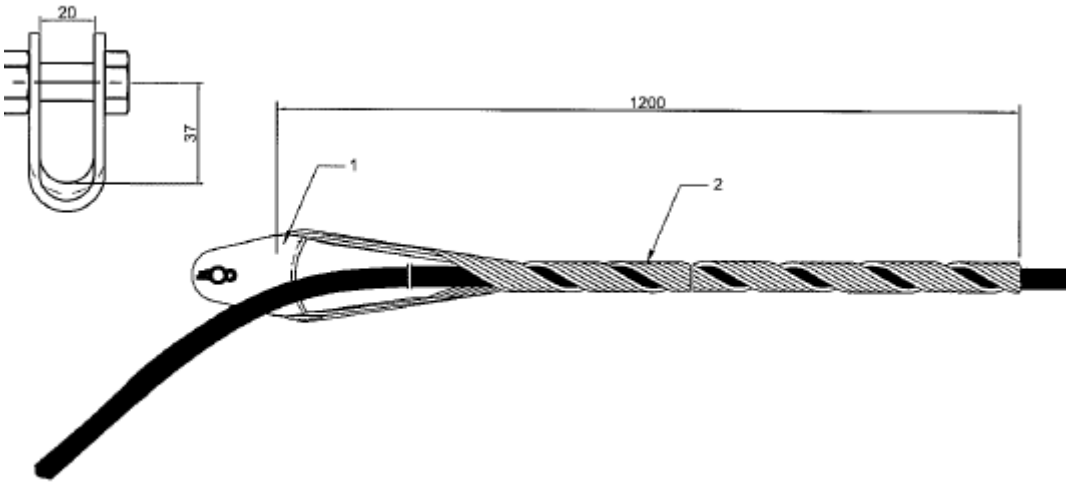


Figura 32. Preforme Vano 200

2.2.27.3. **Preforme De Retención Doble**

Preformado exterior - preformado por hilos de acero revestido de aluminio. Tiene un alto agarre por lo que no extruirá el cable óptico. Los hilos tienen una marca de color para evitar el error de instalación y es conveniente para una rápida instalación e intercambio de cables pre-torcidos.

Preformado interior - preformada por alambre de acero revestido de aluminio. Protege el cable óptico y reduce la vibración. El extremo de los hilos debe estar ligeramente doblado hacia afuera siguiendo la dirección radial para evitar la extrusión del cable óptico. Los hilos se preforman en cuatro subpaquetes para evitar errores de instalación y

para una instalación rápida. Los dos extremos están pintados con marcas de colores para intercambiar el cable.

Figura 33. Preforme Vano 600

2.2.27.4. **Herraje De Suspensión**

- se aplica a ADSS.
- El ángulo de giro de las líneas con una grampa de suspensión simple no debe ser mayor a 25º. El ángulo de
- giro de las líneas con grampas de doble suspensión debe estar entre 25º y 60º.
- El vano recomendable es más de 150 m.
- Este producto solo debe ser utilizado por profesionales capacitados.
- No cambie las cantidades y longitudes de los ensamblajes al azar.
- preformados no se puede usar repetidamente.
- Si tiene alguna duda, póngase en contacto con el departamento de tecnología de nuestra empresa.

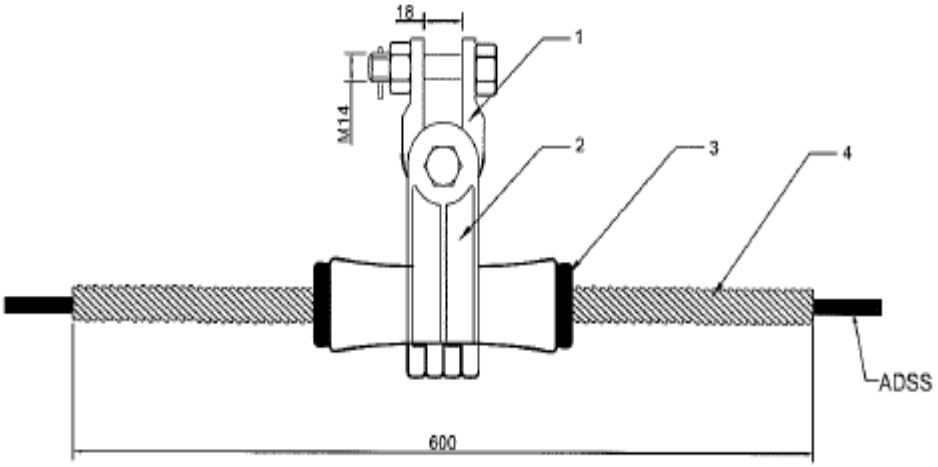


Figura 34. Herraje de suspensión.

2.2.27.5. **Amortiguador**

Para ADSS, el amortiguador de vibraciones en espiral es comúnmente usado. Es un tipo de absorbedor de impacto, es muy efectivo para la vibración de alta frecuencia de líneas de distribución de cable de pequeño diámetro, líneas estáticas y de fibra óptica. El amortiguador de vibraciones en espiral está hecho de plástico de ingeniería de alta resistencia con buena resistencia a los golpes y resistencia al envejecimiento. Está compuesto por una parte de amortiguación y una parte apretada, una parte apretada que sujeta el cable, una parte de amortiguación que disipa la energía de vibración al impactar el amortiguador de vibración en espiral y el cable óptico, para lograr el efecto de reducir la vibración eólica de la línea.

Este tipo de amortiguador de vibraciones es muy efectivo para cables de diámetro pequeño, pero para el cable de diámetro grande, el efecto no es ideal. En una palabra, su efecto de amortiguación de vibraciones depende de la relación entre la masa y la frecuencia del amortiguador de vibraciones y el cable óptico.

La principal razón de su uso generalizado es:

- El principio de la tecnología es maduro, se ha demostrado que es ideal tanto en el laboratorio como en la línea de transmisión, tenemos muchos años de experiencia operativa.
- Estructura simple, fácil de instalar, fácil de mantener.

El amortiguador de vibración en espiral es muy sensible a la vibración de alta frecuencia del cable ADSS; se instala de acuerdo a la siguiente tabla:

Vano (m)	Cantidad
101 - 200	2
200 - 600	4
600 - 2000	6

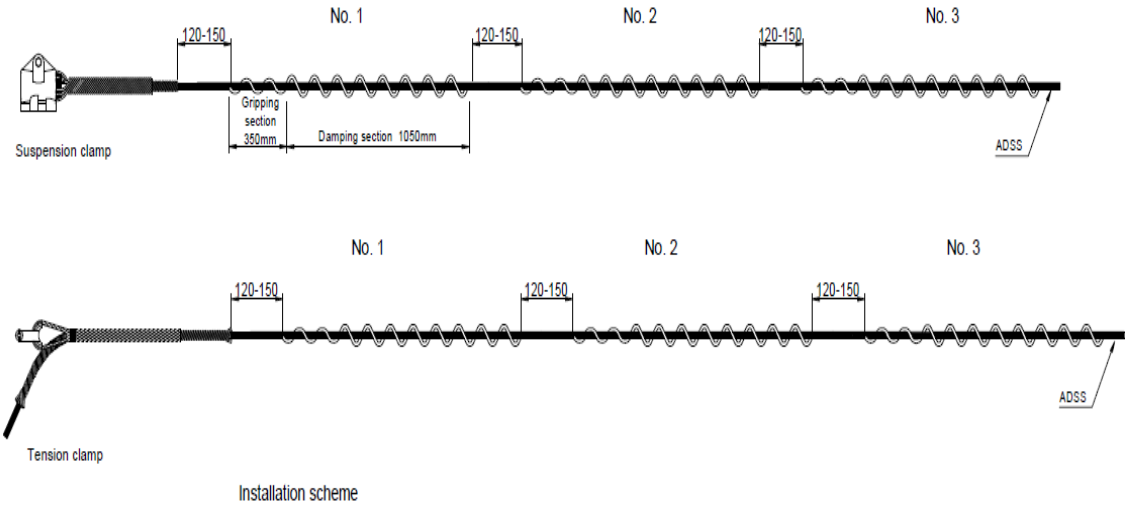


Figura 35. Amortiguador

2.2.27.6. **Retenida**

A continuación, se detalla casos donde se instalan retenidas.

- Las retenidas se deben de instalar antes que se instalen los cables de fibra óptica.
- No se podrá fijar a árboles o plantas ni estructuras de edificios o viviendas.
- En zonas urbanas e interurbanas deberá contar con sus protectores de riostra correctamente señalizada y visible.
- Se debe instalar retenida en cada inicio y fin de línea.
- Cuando exista doble remate, se instalará una retenida de cada lado del poste en dirección opuesta a la tensión de los cables de fibra óptica.
- Cuando exista ángulos de 90 grados se deberá colocar retenidas a cada lado contrarrestando los esfuerzos.
- Toda retenida se deberá instalas con aislador de loza tipo tracción

Figura 36. Retenida instalada.

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1. TIPO Y DISEÑO DE INVESTIGACIÓN

3.1.1. Tipo De La Investigación

Según Sampieri (2014) no se deben considerar los alcances como "tipos" de investigación, ya que, más que ser una clasificación, constituyen un continuo de "causalidad" que puede tener un estudio.

Menciona que visualizar qué alcance tendrá nuestra investigación es importante para establecer sus límites conceptuales y metodológicos. En ese entender nuestro alcance del presente trabajo de investigación fue de tipo exploratorio, descriptivo.

3.1.2. Diseño De Investigación

De acuerdo a la propuesta de Sampieri (2014) la investigación fue de tipo experimental, y Diseño Pre- experimental, lo que significa que existió la intervención del investigador para manipular la variable independiente (arquitectura de detección de objetos), para medir la precisión de la variable dependiente (tipos de ferretería instalada); que está conformado por un solo grupos.

GE X Ox

Donde:

Ge: Grupo Experimental (partes de herraje instalado en estructuras mediante fotos)

X: implementación del modelo

OX: Prueba de salida (determinación de una arquitectura que estime partes de herrajeria)

3.2. POBLACIÓN Y MUESTRA

3.2.1. Población

La población estuvo comprendida por 300 estructuras que son la cantidad de estructuras instaladas en el tramo Nuñoa Macusani donde la unidad de análisis fue de 5 fotografías por estructura desde diferentes ángulos y posiciones de profundidad. Por lo tanto, la población total de fotografías fue de 1500 imágenes.

3.2.2. Muestra

Para la muestra se utilizaron 251 fotografías seleccionadas por conveniencia y a criterio analizando los requerimientos de las redes neuronales convolucionales.

3.2.3. Lugar de Estudio

La investigación se realizó en el año 2020 en el tramo denominado como Nuñoa-Macusani del proyecto regional de banda ancha Puno, Ubicada entre el Distrito de Nuñoa Provincia de Melgar región Puno y el distrito de Macusani de la provincia de Carabaya región Puno; la distancia aproximada es de 142 kilómetros de tendido y más de 400 estructuras a instalarse donde se utilizó grandes cantidades de ferretería para dicha ejecución y se registró todas las fotografías de cada estructura para la presente investigación.

3.3. TÉCNICA E INSTRUMENTO DE RECOLECCIÓN DE DATOS

El tipo de recolección de información fue mediante toma de fotografías y la recolección de los respectivos documentos pertinentes de las instancias correspondientes para identificar correctamente el material instalado se utilizó manuales facilitados por la empresa ejecutora

3.3.1. **Técnicas**

- La recolección de datos consistió en ir al lugar del tramo indicado guiado mediante un archivo kmz donde muestra todas las estructuras instaladas y las que se están utilizando
- Seguidamente se registra las fotografías ítem por ítem mediante un celular que para este caso fue de marca HUAWEI modelo MATE 20
- Una vez conseguido las imágenes se procede a encarpetar y seleccionar las imágenes para formar la data de entrenamiento.
- Luego que se formó la data con las 252 imágenes seleccionadas se procede a etiquetar cada una de las imágenes con el programa llabelling para realizar las cajas de donde están los objetos a entrenar.
- Es así que se logra tener para cada imagen un archivo .XML de donde se detalla las coordenadas de las cajas realizadas en cada imagen.

3.3.1.1. **AUMENTO DE DATOS**

Cuando ya se tiene las imágenes con sus respectivas etiquetas en archivo. XML se procedió a realizar una técnica de data Aumentation, ya que los modelos generalmente requieren gran cantidad de imágenes, por ello que se aplicó esta técnica incrementando

hasta 1482 imágenes etiquetados formando así una carpeta completa para entrenamiento y validación.

3.4. PLAN DE PROCESAMIENTO Y ANÁLISIS DE DATOS

3.4.1. Análisis de datos

Para llevar a cabo la investigación primero tuvimos que buscar toda la información; luego preparamos el entorno de desarrollo que en este caso fue Python y fue desarrollado en Google colab plataforma que trabaja sobre la nube con GPU sus respectivas librerías instaladas,

Para el desarrollo y ejecución de la investigación se utilizaron las siguientes herramientas y materiales:

Laptop con las siguientes características:

- Sistema Operativo: Win10 de 64 bits
- Memoria: 8 Gb.
- Procesador: Intel® Core™ i5.

Entorno de desarrollo de programación:

- Jupyter Notebook (a nivel local)
- Spyder (a nivel local)
- Google Colab (Entorno gratuito de Jupyter Notebook que se ejecuta en la nube con GPUs.)
- Google Drive como plataforma de almacenamiento y nexo con Google Colab.

- Lenguaje de programación: Python versión 3.6

Seguidamente se utilizaron las arquitecturas YOLOV4 y YoloV5 configurando los respectivos parámetros y utilizando las técnicas de transfer learning para empezar con los pesos que viene entrenados en grandes volúmenes de datos:

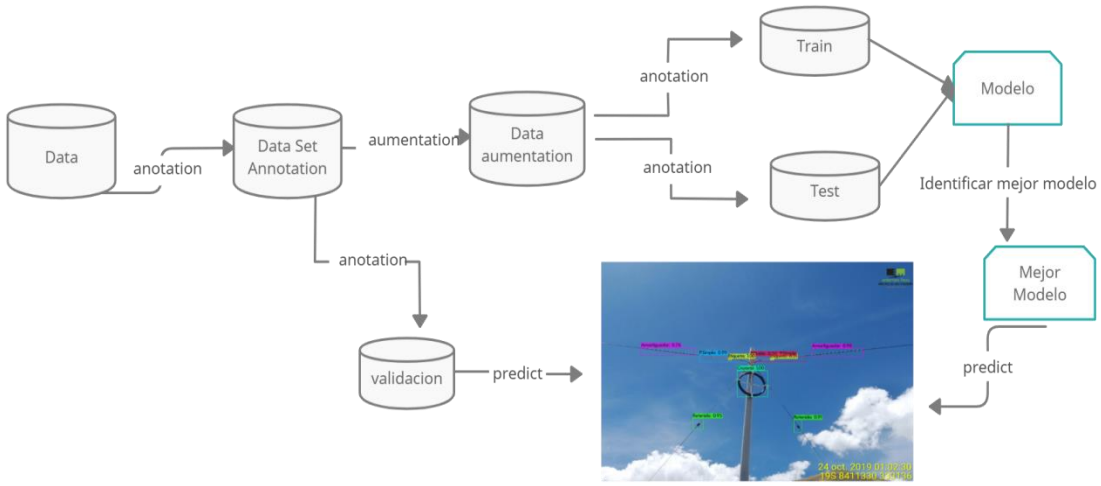


Figura 37. Diagrama de construcción de modelos de detección objetos.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

Para la presentación de resultados se presenta en primer lugar la arquitectura de YoloV4 en darknet, seguido de la arquitectura YoloV5 en PyTorch. Se elige en este orden por la fecha de desarrollo y publicación de cada arquitectura mencionada, asimismo en cada una de ellas se detalla los parámetros utilizados y el resumen de los resultados y gráficos.

4.1. PREPARACIÓN DE DATA

Para la preparación de data se utilizó la cantidad de 271 imágenes de muestra de los cuales 20 imágenes de estas se separó en una carpeta denominada VALIDACIÓN, estas 20 imágenes no se utilizaron en el modelo de entrenamiento ni de test, fue utilizado para las predicciones finales de estimación del modelo, por lo consiguiente separando la carpeta de validación se tuvo 251 imágenes que se les aplico la técnica de data augmentation logrando obtener la cantidad de 1482 imágenes, estos fueron distribuidos aleatoriamente en el 80% para entrenamiento equivalente a 1199 imágenes y el 20% para test equivalente a 283 imágenes.

Tabla 1. División de conjunto de imágenes de herraje instalado en estructuras del tramo Nuñoa-Macusani.

CONJUNTO DE DATA	CANTIDAD IMÁGENES	% DE IMÁGENES
Train Data	1199	80.0%
Test Data	283	20.0%
Valid Data	20	---
Total	1502	100%

Fuente: Elaboración propia a partir de los datos.

Tanto para las arquitecturas de YoloV4 y YoloV5 se utilizó la GPU de Google Colab teniendo las siguientes características:

4.1.1. **Parámetros utilizados Yolo V4 Darknet**

Tabla 2. Parámetros configurados para entrenamiento en Yolo V4

Parámetros	Valor
tamaño de imágenes de entrada	512*512
Batch Size	64, 32
subdivisión	32, 32

Fuente: Elaboración Propia a partir de resultados

4.1.2. Resultados de entrenamiento con diferentes parámetros

Tabla 3. Resumen de pruebas con diferentes parámetros en YoloV4

Pruebas	epoch	Batch Size	tiempo train(h)	mAP %	Precision	Recall	F1-Score
1	2000	64	3	63.77	0.72	0.64	0.68
2	7000	64	6	79.58	0.82	0.82	0.82
3	9700	64	10	84.16	0.86	0.87	0.87
4	11300	64	12	88.78	0.89	0.89	0.89
5	14000	64	16	89.45	0.90	0.90	0.90
6	7371	32	7	47.37	0.75	0.45	0.56

Fuente: Elaboración Propia a partir de resultados

De la tabla anterior se observa que se realizó 6 tipos diferentes de entrenamiento variando los parámetros donde se observa que se empezó a analizar con 2000 épocas con batch size de 64, para lo cual el tiempo de entrenamiento fue de 3 horas aproximadamente y se tuvo mAP de 63.77%, precisión de 72%, recall de 64% y F1-Score de 0.68%.

Para estos resultados a un inicio parecen regulares en la precisión, por ende, se continua con el análisis de posteriores resultados para 7000 épocas, 9700 épocas y 11300 épocas

donde se llegó en esta última a tener mAP de 88.78%, precisión de 89%, recall de 89% y F1-score de 89% donde indican según la tabla anterior que fueron los mejores resultados hasta esa época.

Cumpliendo lo recomendado en la arquitectura darknet de yoloV4, donde indica que las épocas correctas para el entrenamiento son :" *clases*2*1000"* es por ello que al tener 7 clases en nuestro modelo se entrenó durante 14000 épocas y es donde ahí se logra el mAP más alto con respecto a las demás pruebas, después de 16 horas de entrenamiento, se alcanza obtener el 89.45% de objetos en promedio detectados (mAP) mediante el modelo construido, con la métrica de precisión se obtuvo el 90%, la métrica de recall 90% y F1-Score de 90%.

Asimismo se realizó una sexta prueba, configurando a 14000 épocas, modificando el batch size de 64 a 32, entrenando durante 7 horas y estando en la época 7371 se decide paralizar el entrenamiento ya que se notó que los parámetros de medición no incrementaban, es por ello que se evidencia en la anterior tabla que en la sexta prueba se tiene un mAP de 47.37% de los datos, y comparando con la segunda prueba que a las 7000 épocas con batch size de 64 ya se tenía un mAP de 79.58%, lo cual se verifica que mientras el batch size sea mayor, los parámetros de medición tienden a mejorar.

Indicar también que se intentó probar configurando el batch size a 128, lo cual después de la segunda época iniciada se tiene colapso de memoria RAM asignada y se paraliza automáticamente; es por ello que todas las pruebas se realizaron con el tamaño de lote de 64 (batch size).

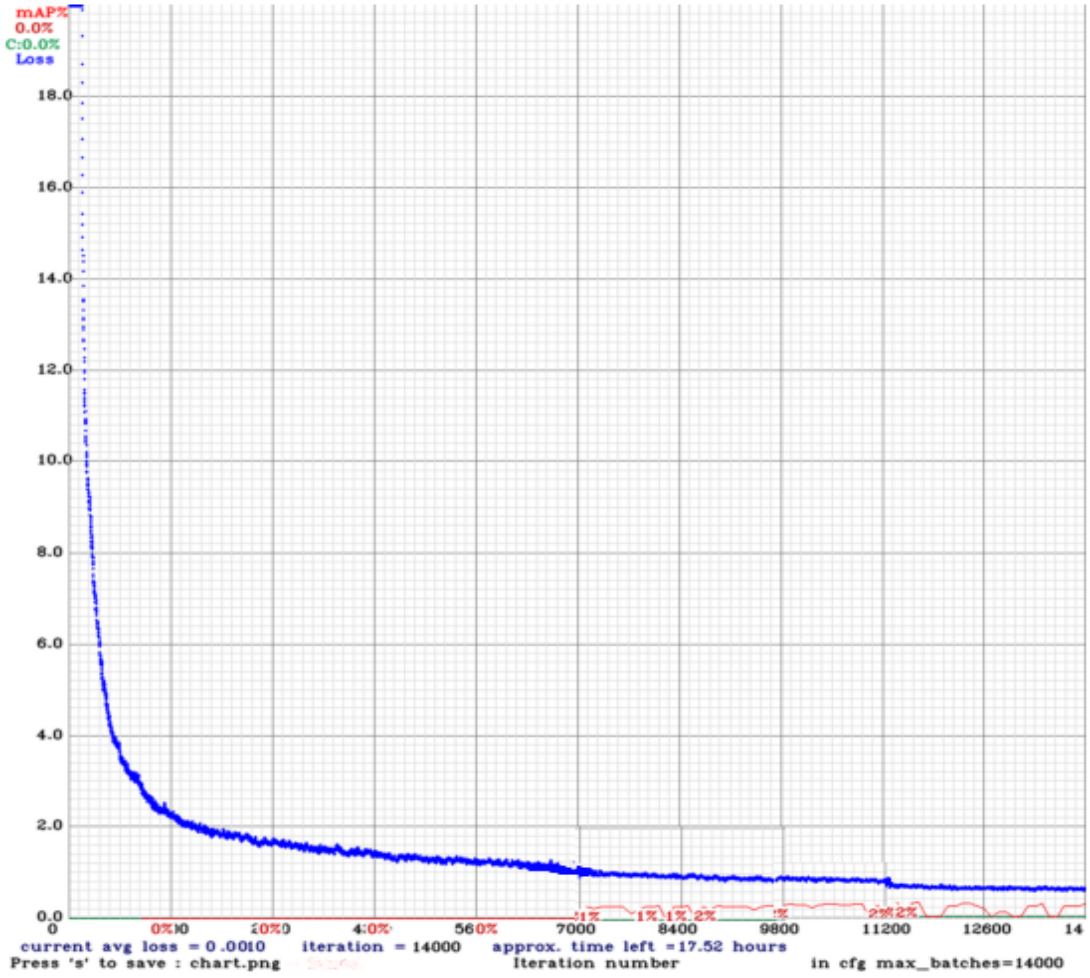


Figura 38. Gráfico de pérdida del modelo que tuvo mejores métricas.

Del grafico anterior se evidencia la tasa de error o perdida en el entrenamiento durante las 14000 epocas de entrenamiento y a medida que va pasando cada epoca va disminuyendo esta perdida, se muestra que a partir de la epoca 8400 empieza a ser menor a 1 la perdida y al llegar a las 14000 epocas se tiene menos de 0.5 de perdida.

A continuación, se presenta el resumen del modelo detallado de la 5ta prueba que fue la de mejores métricas de evaluación en esta arquitectura.

Tabla 4. Resultados de prueba 6 que tuvo mejores métricas

Class_ID	Name	AP
0	Amortiguador	93.42%
1	Cruceta	100.00%
2	Etiqueta	63.53%
3	P.Doble	96.45%
4	P. Simple	96.29%
5	Retenida	88.22%
6	Suspensión	88.24%
	TOTAL	89.45%

Fuente: Elaboración propia a partir de resultados

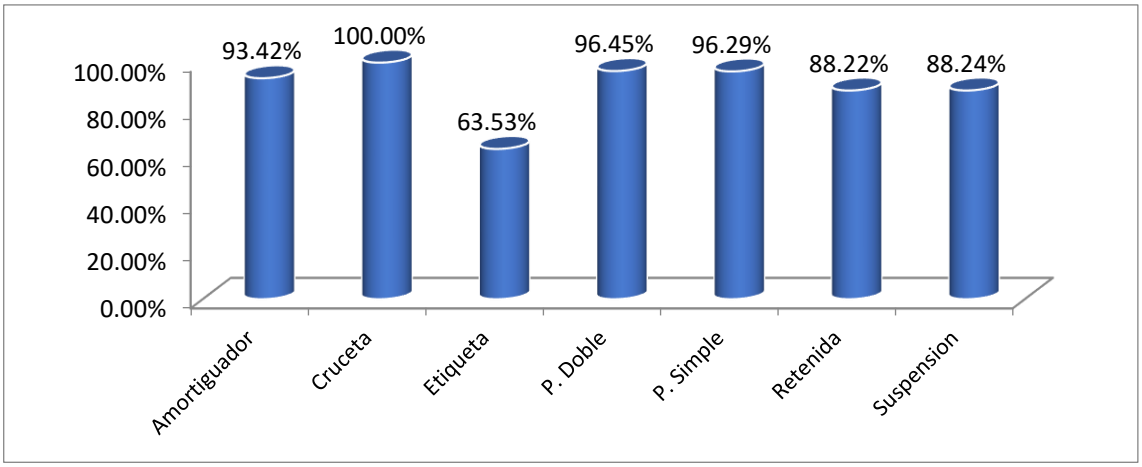


Figura 39. Resultados de AP en la arquitectura de mejores métricas.

Del cuadro y grafico anterior se observa que los indicadores de la arquitectura son los mismos que se detalla en la tabla número 05 de la prueba 06, también se muestra las detecciones detalladas de la carpeta test, donde en general identifico 2031 objetos en todas las imágenes dadas de los cuales identificó 1341 verdaderos positivos, 154 objetos de falsos positivos y 150 objetos de falsos negativos.

Seguidamente se detalla que el objeto amortiguador tuvo un Ap de 93.42%, la cruceta tuvo Ap del 100%, las etiquetas tuvieron una precisión promedio de 63.53%; el más

bajo de todos los AP, esto resultado se da porque podría haber sido por el tamaño del contorno del objeto que es muy pequeño, seguidamente el objeto Preforme Doble con 96.5% de AP, el preforme simple con AP de 96.29%; las últimas dos precisiones promedio son las retenidas y las suspensiones con 88.22% y 88.24% de AP respectivamente.

4.1.3. **Predicción**

A continuación se presenta las predicciones realizadas en imágenes que no fueron utilizados para entrenamiento ni para test ya que estas fueron separadas en la carpeta de validación y no es utilizaron para la construcción del modelo deep learning, cabe precisar que el modelo aprendió con imágenes de entrada de 512*512, pero para la predicción se puede utilizar cualquier tamaño de imagen como entrada.

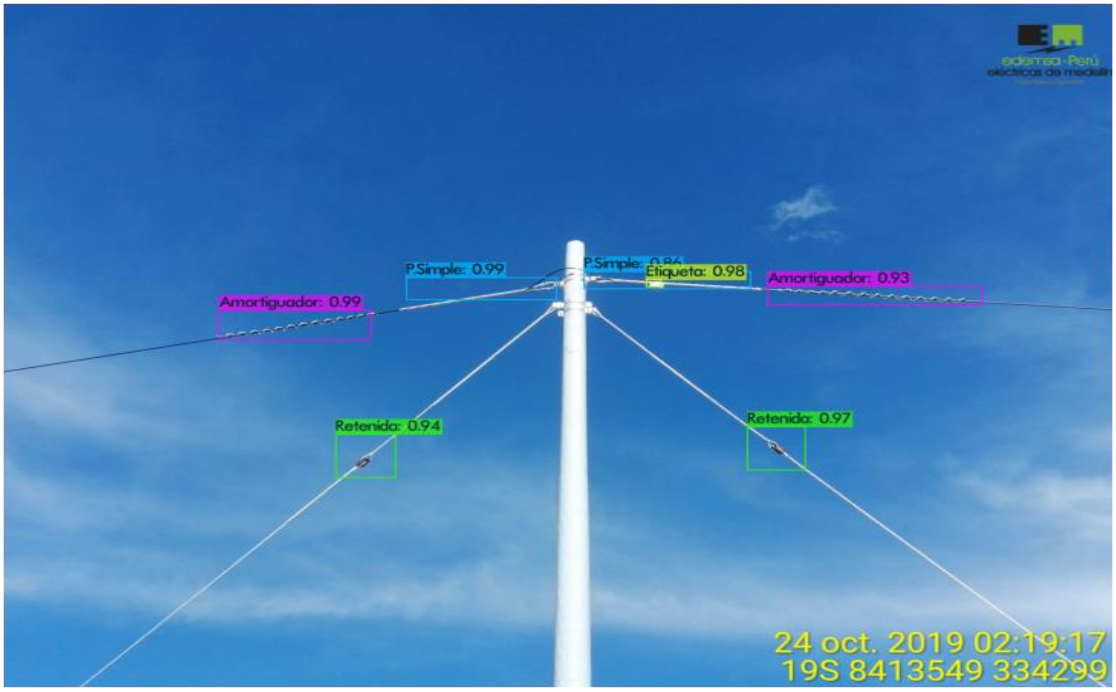


Figura 40. Predicción en imágenes nueva

Figura 41. Predicción en imágenes nuevos

De las imágenes número 43 y 44 que fueron predichos por la arquitectura entrenada de la carpeta validación de imágenes, se muestra el 100% de detecciones correctas, lo que se debe indicar en estas imágenes es que fueron tomadas directamente a todo el herraje instalado y una buena iluminación.

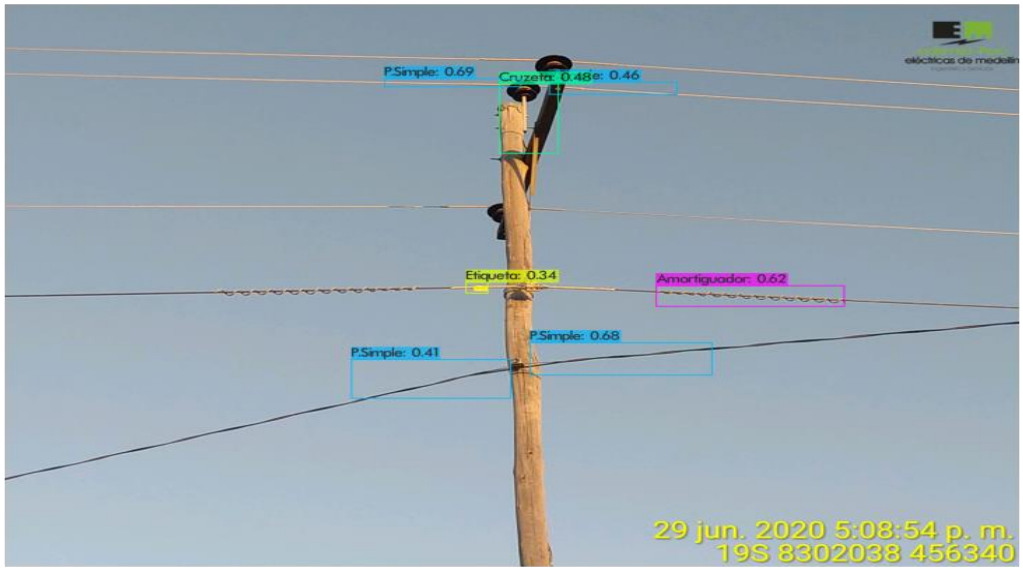


Figura 42. Predicción en imágenes nuevos.

Utilizando la arquitectura de entrenamiento que tuvo las mejores métricas se pone a prueba la siguiente imagen donde detecta objetos de más que son ajenos al estudiado, esto se justifica por el hecho de que las imágenes de entrenamiento son para exclusivamente postes nuevos instalados que no tienen otro cable más que solo el de fibra óptica.

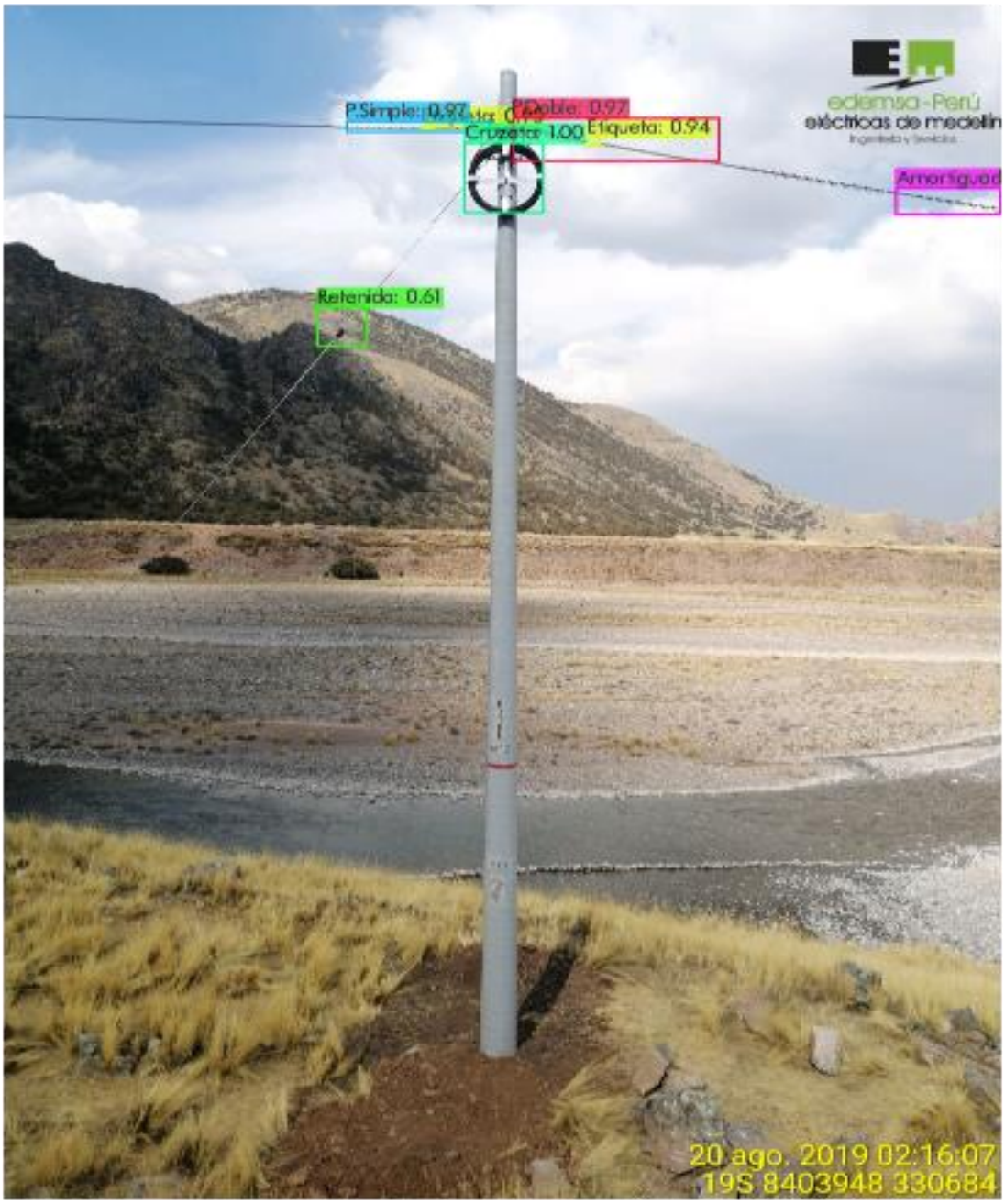


Figura 43. Predicción en imágenes nuevos.

De la imagen anterior se evidencia que con una foto bien tomada desde la frontal y viéndose desde la base, la arquitectura deep learning implementada logra detectar correctamente todas las partes de herraje instalado menos un amortiguador.

4.2. ARQUITECTURA YOLOV5 PYTORCH

4.2.1. Parámetros utilizados en YoloV5 PyTorch

Tabla 5. Parámetros configurados en el entrenamiento de YoloV5

Parámetros	Valor
tamaño de imágenes de entrada	640*640
Batch Size	8, 16
Subdivisión	32, 32

Fuente: Elaboración Propia a partir de resultados

4.2.2. Resultados de entrenamiento con diferentes parámetros

Tabla 6. Métricas para diferentes pruebas en la arquitectura YoloV5.

Pruebas	epoch	Batch Size	tiempo train(h)	mAP	Precision	Recall
1	50	8	2	0.583	0.389	0.71
2	100	8	4	0.675	0.487	0.741
3	150	8	6	0.678	0.553	0.763
4	199	8	8	0.67	0.558	0.755
5	20	16	1	0.804	0.83	0.813
6	50	16	2	0.927	0.929	0.91
7	99	16	4	0.953	0.953	0.945

Fuente: Elaboración Propia a partir de resultados

De la tabla anterior se deduce que se analizó para batch size de 8 en 04 pruebas diferentes y 3 pruebas en diferentes épocas para bath size de 16; de los cuales en la primera prueba después de entrenar durante dos horas se logra tener un mAP de 58.3%, precisión de 0.389 y recall de 0.71. Después de 100 épocas en la prueba 03 se aprecia que se logra tener un mAP de 67.8%, precisión de 55.3% y recall de 76.3%. Se paraliza este bloque de entrenamiento a las 199 épocas porque se logra apreciar que el mAP no varía y por el contrario disminuye a 67%, la precisión incrementa ligeramente a 0.558 y el recall disminuye a 0.755.

Luego de paralizar el entrenamiento en 199 épocas del grupo de batch size de 8, se incrementa el batch size a 32, lo cual a las 3 épocas se paraliza el entrenamiento automáticamente por colapso de memoria RAM, por ende se configura el batch size a 16 y con el tamaño de entrada de imagen de 614 se empieza a analizar luego de una hora de entrenamiento en la época 20 obteniendo un resultado muy positivo del mAP de 80.4% superando a las 199 épocas del anterior bloque de entrenamiento.

Es así que se deja entrenar por 99 épocas y se analiza los resultados donde se puede apreciar de la tabla anterior que luego de 4 horas de entrenamiento se logra un mAP de 0.953, la misma métrica de precisión de 0.953 y el recall de 0.945, indicándonos así que este es la arquitectura que tiene las mejores métricas de entrenamiento de la arquitectura YoloV5 de PyTorch.

Tabla 7. Resultados obtenidos de la prueba que tuvo mejores métricas de evaluación en YoloV5 PyTorch.

Class_ID	Name	AP
0	Amortiguador	99.30%
1	Cruzeta	99.50%
2	Etiqueta	82.50%
3	P.Doble	98.50%
4	P.Simple	98.20%
5	Retenida	94.50%
6	Suspension	94.80%
	TOTAL	95.33%

Fuente: Elaboración Propia en base a resultados.

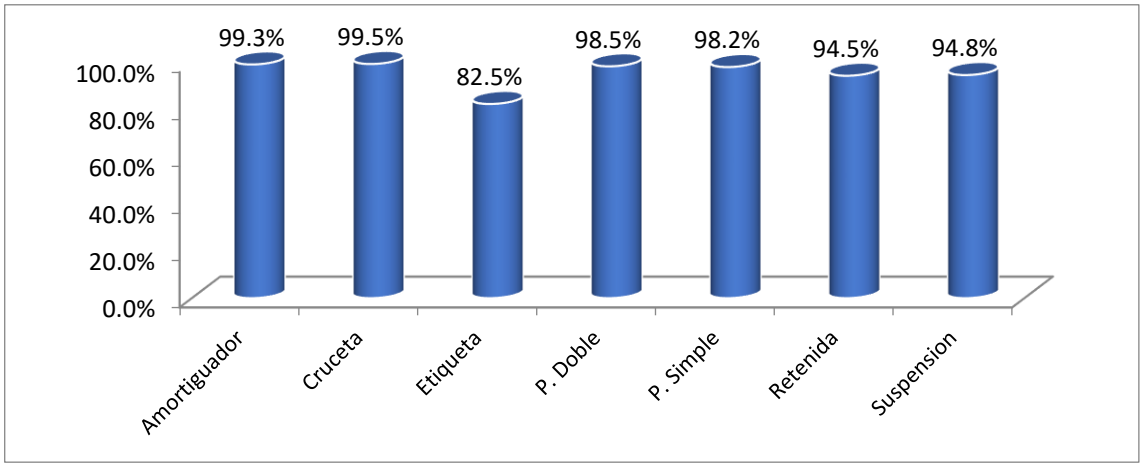


Figura 44. AP de los objetos detectados de mejores métricas en YoloV5

Del cuadro y grafico anterior se deduce que de la última arquitectura entrenada, los objetos identificados como amortiguador fueron estimados correctamente en el test el 99.3%, seguido del objeto cruceta que del 100% de los existentes, el 99.5% fueron detectados correctamente, seguido de las etiquetas con el 82.5%, en el objeto preforme doble el 98.5% fueron detectados correctamente, el objeto preforme simple el 98.2% fueron detectados correctamente, las retenidas el 94.5% identificados correctamente, y finalmente el herraje de suspensión del total fueron el 94.5% detectados correctamente.

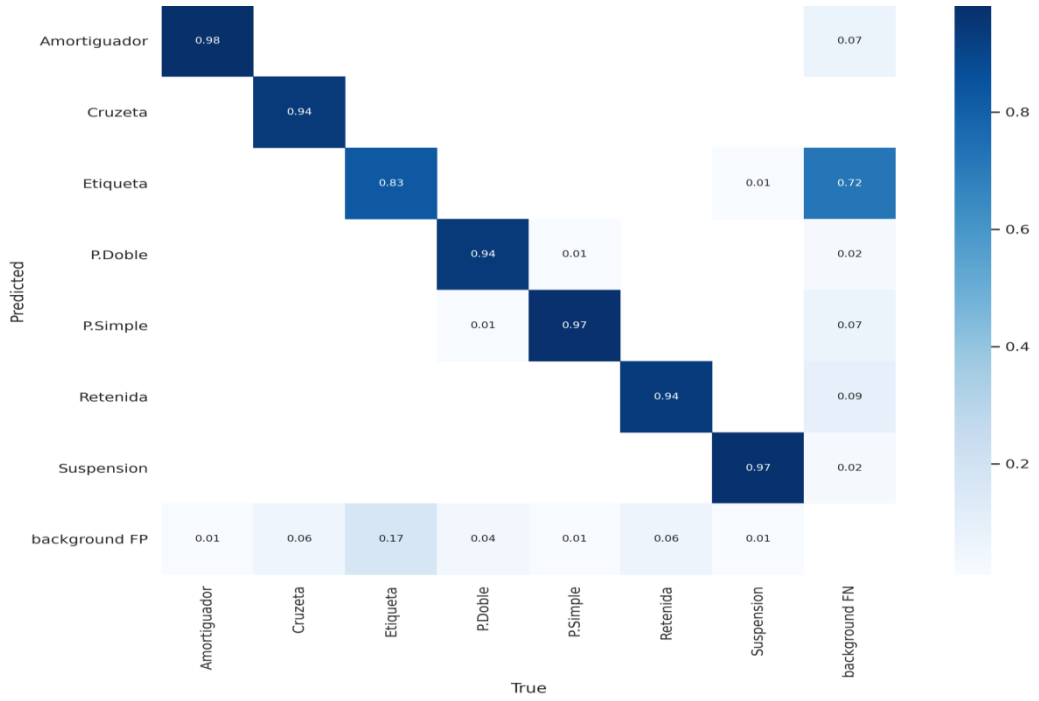


Figura 45. Mapa de Confusión de los AP de la prueba de mejores métricas.

Del grafico anterior se describen los resultados de matriz de confusión donde se incrementa una fila de fondo con falso positivo y se identifica que el objeto clasificado como amortiguador tuvo 0.01 del total como falso positivo, seguido de cruceta que tuvo el 0.06 de falso positivo, continuado de las etiquetas con 0.17 de falso positivo, seguido del preforme doble con 0.04 de falso positivo, luego el preforme simple tuvo 0.01 de falso positivo las retenidas de 0.06 de falso positivo y finalmente el herraje de suspensión que tuvo el 0.01 de falso positivo.

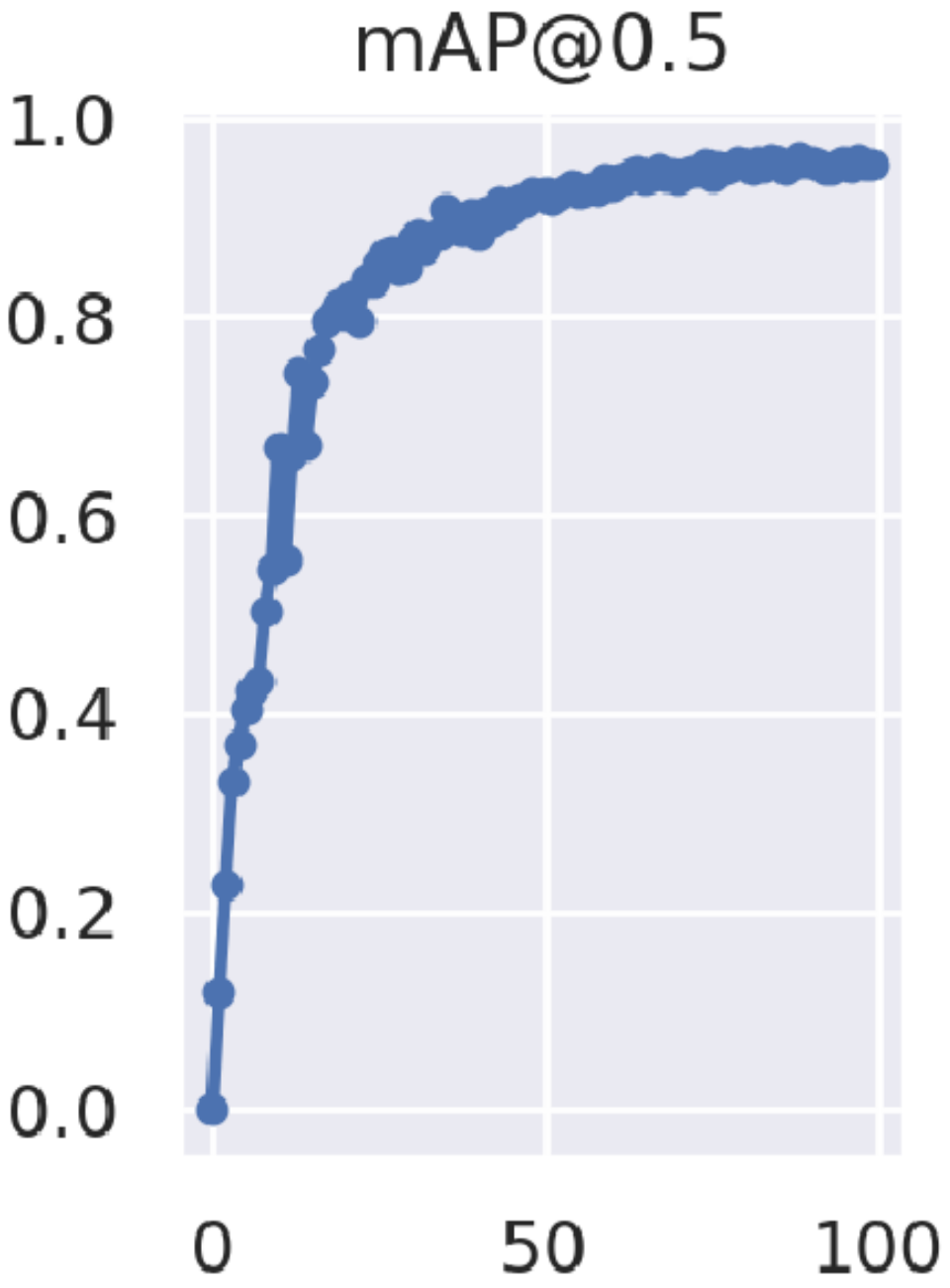


Figura 46. Evolución de mAP en las 99 épocas de entrenamiento.

Del anterior grafico se evidencia como va incrementando el valor mAP(Media) a medida que van pasando las épocas de entrenamiento y se va observando que a partir de la época 80 de entrenamiento va permaneciendo constante y ya no se nota una mejora significativa en cuanto a esta métrica de entrenamiento.

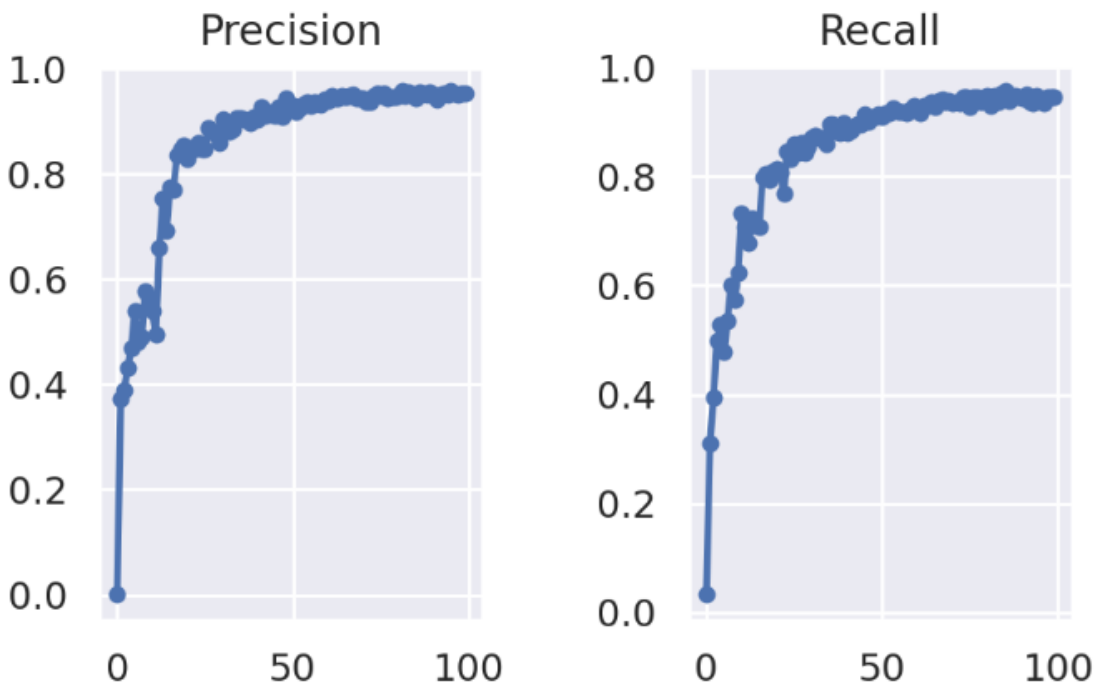


Figura 47. Evolución de Precisión y Recall en las 99 épocas de entrenamiento

El grafico anterior muestra las métricas de evaluación de precisión y Recall en función vayan aumentando las épocas de entrenamiento y se observa que estas métricas tienden a converger desde la época 70 en adelante y se estima que empieza a sobre ajustarse desde esta época, por ende se concluye que las 100 épocas utilizadas en el entrenamiento son las mejores métricas que puede dar el modelo deep learning

4.2.3. **Predicciones realizadas con la arquitectura**

A pesar de que el modelo deep learning aprendió a detectar imágenes de entrada de tamaño 640*640, para la predicción se puede pasar el tamaño real de la imagen es decir de cualquier otro tamaño y la arquitectura será capaz de detectar con estos tamaños diferentes de imágenes.

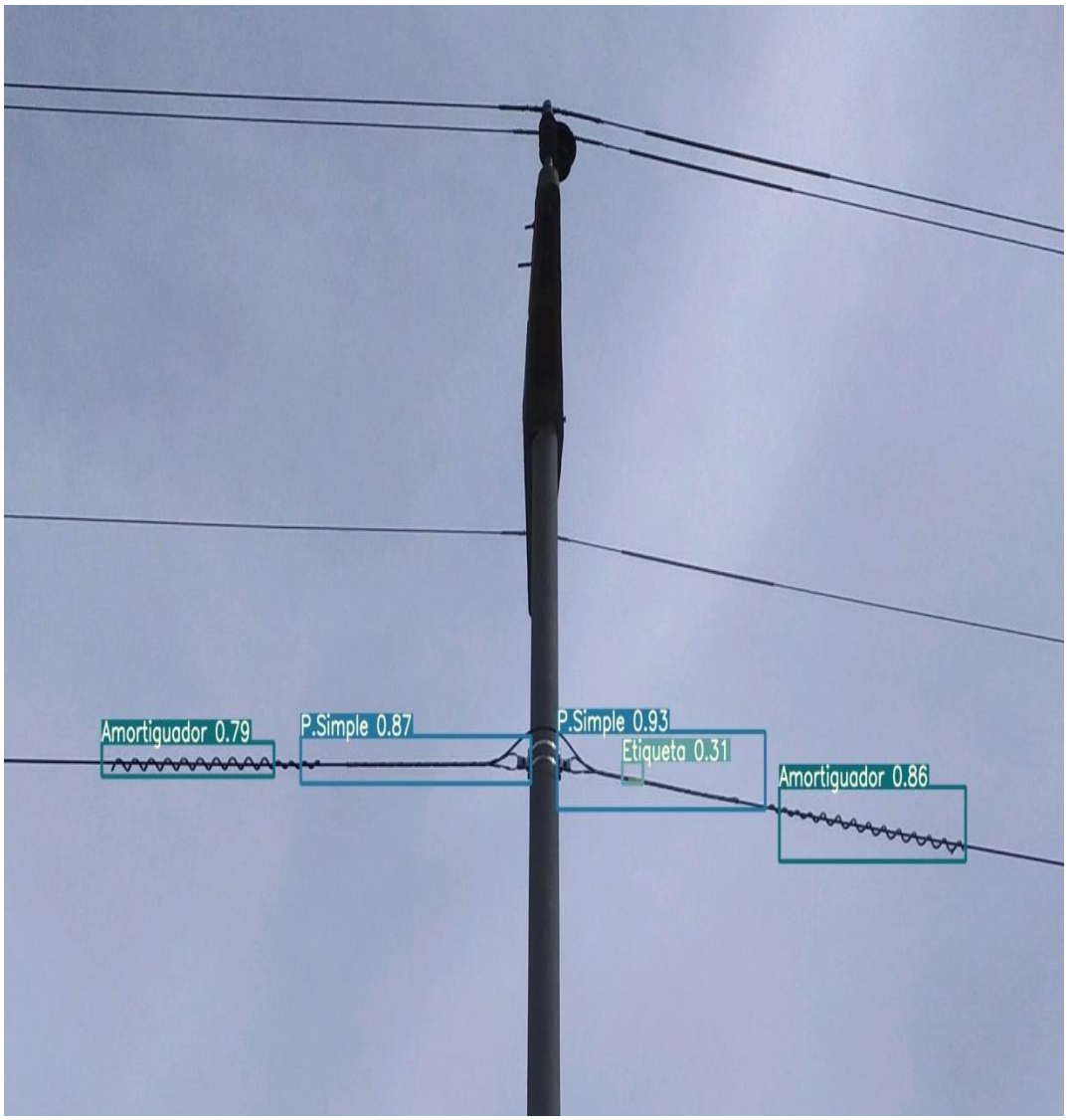


Figura 48. Predicción en imagen nueva.

Del grafico anterior se observa la estimación realizada con la arquitectura desarrollada en las imágenes de validación, donde se aprecia que a pesar de ser un poste existente que contiene cable de media tensión no lo detecta como parte de la arquitectura y todas las partes de la herrajeria instalada que retiene el cable de fibra óptica.

Figura 49. Predicción en imagen nueva

En este experimento deep learning se detecta todos los objetos entrenados en la arquitectura, excepto la etiqueta ya que si se evidencia la estructura se da cuenta de que efectivamente no cuenta con etiqueta dicha estructura.

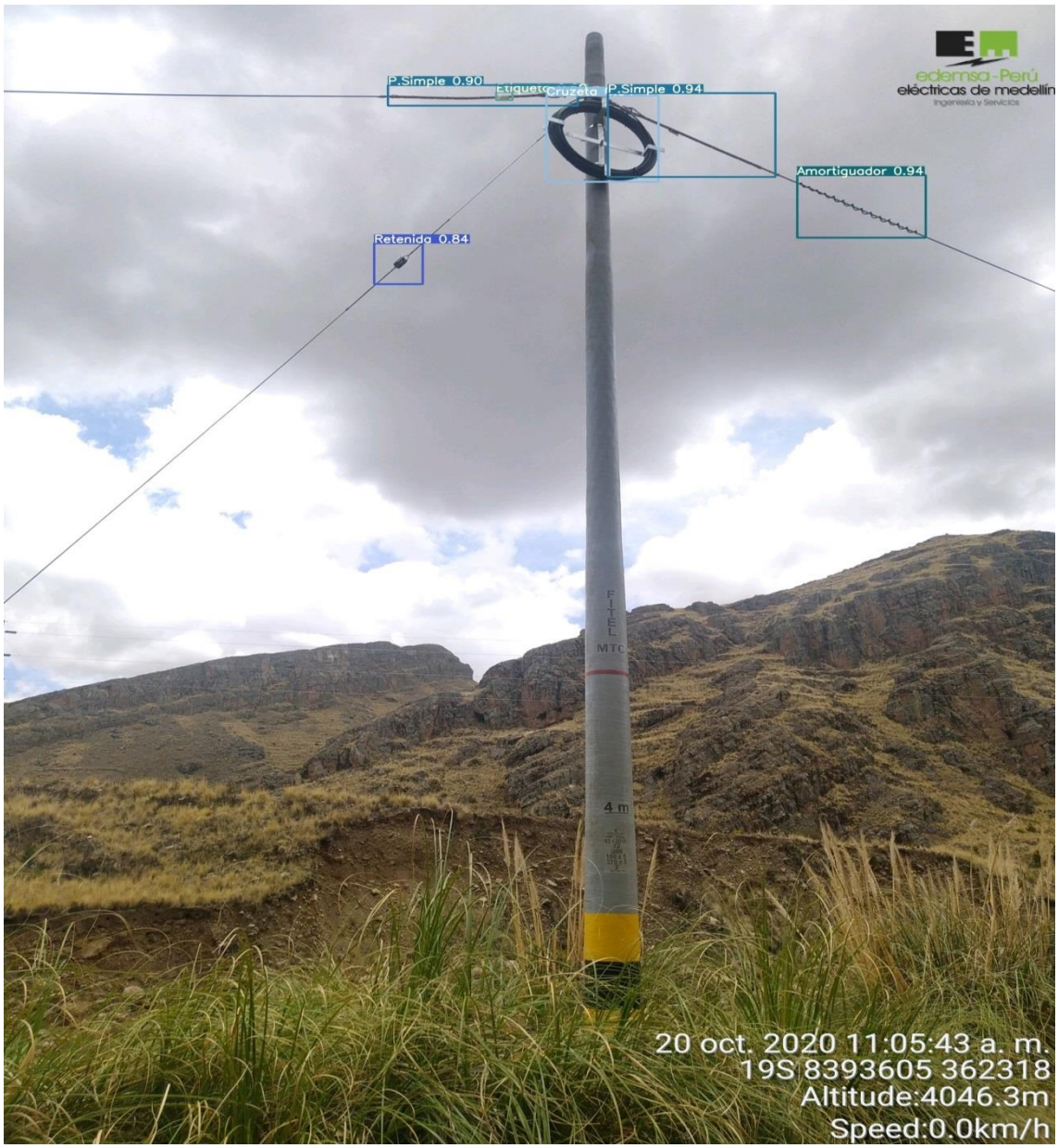


Figura 50. Predicción en imagen nueva.

Se le pasa a la arquitectura una foto de toda la estructura desde la base y se muestra la detección de todos los objetos excepto una etiqueta que si está instalada pero no fue detectada, en este caso hubo un verdadero negativo que no fue identificado, bajo esta perspectiva se indica que el modelo detecta correctamente cuando la imagen es tomada desde una buena ubicación respecto a iluminación y que sea la parte solo del herraje instalado, es decir con un zoom hacia los objetos.

4.3. DISCUSIONES

En la tesis de Gutiérrez(2019), sobre detección de armas en videos mediante técnicas de deep learning utiliza la arquitectura de yoloV2 donde logra obtener el 71% de acierto, donde realizando la comparación con nuestra implementación, no se acerca a los resultados que obtuvimos con la arquitectura de yoloV4, asimismo estos resultados es mucho menor con la arquitectura de YoloV5 que obtuvimos mejores métricas; por ende en el momento de desarrollo de esa tesis aun solo existía la arquitectura de YoloV2 a lo mucho y por ello obtiene dichas métricas de evaluación, además de ello mencionar que las pruebas lo hicieron mediante videos, y en la presente investigación fue mediante imágenes en su totalidad; al final concluye que la arquitectura yoloV2 es más rápido que las demás utilizadas en esa investigación.

En la tesis sobre detección de objetos aplicada a vehículos autónomos con técnicas deep learning utilizo la arquitectura de Faster RCNN para detectar peatones y alcanzo valores de mAP del 82% del mejor de todos los modelos deep learning que implementó, indicar que esta arquitectura no es muy óptima para la detección de objetos a comparación desde la arquitectura YoloV3 es quizá por ello el mAP obtenido en esa tesis y la diferencia que existe con la presente investigación ya que la presente arquitectura entrenada del YoloV5 tuvo el 99% de mAP por lo que podemos mencionar que los resultados obtenidos en esta investigación son bastante positivos, pero cabe indicar que no se utilizan los mismos conjuntos de datos ni la misma arquitectura, pero a pesar de ello las métricas son buenas comparaciones que se realizan.

En la tesis reconocimiento de eventos anómalos en videos de cámara de vigilancia utilizando redes convolucionales propuso tres arquitecturas YOLO, Región Proposal +

CNN(R-CNN) y el SSD encontrando al final un acierto del 86 %, 85% de precisión y 73% de recall; estos resultados si son parecidos a los obtenido en la arquitectura de Yolo V4, pero si comparamos con la arquitectura YoloV5, entonces existen bastante diferencias en los indicar dores de evaluación; por ende se justifica que los resultados obtenidos en la presente investigación fueron tan altos y significativos porque se utilizó la última versión de la arquitectura Yolo V5 donde indica en su teoría que tiene los más altos valores de métricas comparados actualmente con cualquier otra arquitectura existente de detección de objetos.

V. CONCLUSIONES

PRIMERA: En conclusión, se desarrollaron 13 algoritmos diferentes (YoloV4, YoloV5) de detección de objetos para los herrajes instalados en estructuras del tendido de Fibra óptica del tramo Nuñoa-Macusani, el primer proceso fue el etiquetado de datos mediante Llabelling en formato pascalVOc, seguidamente se aplicaron técnicas de data aumentation llegando a tener un total de 1482 imágenes donde para ambas arquitecturas fueron divididos el conjunto de datos en 80% para entrenamiento y 20% para test.

SEGUNDA: En la arquitectura YOLOV4 se realizaron 6 pruebas de estos 5 fueron con tamaño de lote(batch size) 64 y uno de 32; asimismo para la arquitectura YOLOV5 fueron analizados en 7 etapas diferentes donde 4 pruebas fueron con batch size de 8 y las tres últimas pruebas con batch size de 16.

Para la primera arquitectura de yoloV4 se utilizaron imágenes de entrada de 512*512 donde los mejores resultados que se llegaron a obtener fueron en 14000 épocas de entrenamiento con batch size de 64 y el tiempo de entrenamiento fue de 16 horas aproximadamente, logrando a obtener mAP de 89.45%, la precisión y recall que se obtuvo fue del 90%.

TERCERA: Para la segunda arquitectura que fue YoloV5 se tuvieron 7 pruebas en diferentes parámetros de los cuales la mejor fue la séptima prueba donde se entrenaron durante 99 épocas con batch size de 16 y el tiempo aproximado de entrenamiento para esta prueba únicamente fue de 4 horas donde se logró obtener un mAP de 95.3% precisión de 95.3% y Recall de 94.5%, siendo esta la que tuvo las mejores métricas comparando con todas

las pruebas realizadas tanto en YoloV4 y YoloV5, era de esperarse este resultado por ser la arquitectura YoloV5 la última en implementarse y tener las mejores métricas de evaluación según su teoría presentada.

CUARTA: Se realizó estimaciones de detecciones en el grupo de imágenes de validación donde estas imágenes no fueron utilizados para entrenamiento ni para test, y se infiere en las mejores pruebas de cada arquitectura implementada, donde se detecta correctamente los herrajes instalados en las estructuras siempre y cuando estas imágenes sean tomadas desde una buena posición con luz y zoom donde solo muestre el herraje; mientras que si se pasa a la arquitectura imágenes desde la base de la estructura no se llega a detectar en su totalidad los herrajes instalados como las etiquetas que son donde se tuvo la menor precisión.

VI. RECOMENDACIONES

PRIMERA: Para los presentes datos se pretende probar con otras arquitecturas que vayan apareciendo, utilizar diferentes parámetros de entrada en tamaños de imágenes, tamaños de lote, iteraciones entre otros para obtener diferentes y mejores métricas de evaluación ya que en la presente no se pudo desarrollar por la falta de procesamiento GPU propia.

SEGUNDA: Con la presente investigación solo se desarrolló la arquitectura para realizar detecciones de herraje instalado en estructuras de cable de fibra óptica, por ende se recomienda continuar con la línea de investigación y realizar un sistema de detección completo que genere reportes automáticos a través de las detecciones.

TERCERA: Asimismo, se deja la presente investigación como antecedente para futuras investigaciones en este campo y realizar detección de objetos en cualquier campo ya sea de salud, seguridad, sector agropecuario, entre otros, y así reducir el costo de personal, costo de logística, equipos, entre otros.

VII. REFERENCIAS BIBLIOGRÁFICAS

Arriola, I. (2018). *Detección de objetos basada en Deep Learning y aplicada a vehículos autónomos*. Universidad del país Vasco.

Bochkovskiy, A., Wang, C., & Liao, H. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. https://github.com/AlexeyAB/darknet

Calvo, D. (2018). Red Neuronal Recurrente – RNN. *9 Diciembre, 2018*, 1–5. http://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

Castro, W. A. Z. (2020). *Detección y clasificación de células normales de la sangre periférica usando aprendizaje profundo*. Universitat Oberta de Catalunya.

Ccari, A. S. (2019). *DETECCIÓN DE EVENTOS INUSUALES EN IMÁGENES Y VIDEO DE CÁMARAS DE VIGILANCIA*.

Chino, R. C. (2019). *DISEÑO DE UN ALGORITMO DE PROCESAMIENTO DE IMÁGENES DEL SISTEMA DE PESAJE PARA EL CONTROL AUTOMÁTICO DE UNA FAJA TRANSPORTADORA EN LA UNIDAD MINERA MALLAY*. Universidad Nacional del Altiplano.

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV*. http://www.amazon.com/Practical-Introduction-Computer-Imaging-Technology/dp/1118848454/ref=sr_1_6?s=books&ie=UTF8&qid=1415059357&sr=1-6&keywords=opencv

Deng, L., & Yu, D. (2013). Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, *7*(3–4), 197–387. https://doi.org/10.1561/2000000039

Erroz, D. A. (2019). *Visualizando neuronas en Redes Neuronales Convolucionales* [universidad publica de navarra]. https://academica-e.unavarra.es/xmlui/bitstream/handle/2454/33694/memoria_TFG.pdf?sequence=1&isAllowed=y

García, B. N. (2015). *Implementation of Deep Learning Techniques*. UNIVERSIDAD DE LA LAGUNA.

Garralda, L. A. (2018). *Localización de objetos en imágenes mediante técnicas de aprendizaje profundo* [Universidad de la Rioja]. https://biblioteca.unirioja.es/tfe_e/TFE003098.pdf

Gutierrez, C. L. (2019). *E . T . S . de Ingeniería Industrial , Informática y de Telecomunicación Detección de armas en vídeos mediante técnicas de Deep Learning Grado en Ingeniería Informática Trabajo Fin de Grado Resumen*. Universidad Publica de Navarra.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*.

Hernández García, R., García Reyes, E., Ramos Cózar, J., & Guil Mata, N. (2017). *LIBROS_VHDL/Brock J. LaMeres-Introduction to Logic Circuits and Logic Design with Verilog-Springer (2017).pdf humanas en video: estado del arte Features representation models for human actions classification in video: 8*(4), 2227–1899. http://rcci.uci.cu

Huaman, J. M. C. (2019). *DETECCION DE ARMAS DE FUEGO UTILIZANDO REDES CONVOLUCIONALES PROFUNDAS*. Universidad Nacional de San

Agustiın de Arequipa.

Huang, G., Liu, Z., Research, F. A., & Weinberger, K. Q. (2018). *Densely Connected Convolutional Networks*.

Hurwitz, J., & Kirsch, D. (2018). Approaches to machine learning. In *Journal of the American Society for Information Science* (Vol. 35, Issue 5). https://doi.org/10.1002/asi.4630350509

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2007). ImageNet Classification with Deep Convolutional Neural Networks. *Handbook of Approximation Algorithms and Metaheuristics*, 1–1432. https://doi.org/10.1201/9781420010749

LAMEGO, J. A. C. Z. (2017). *Desarrollo de un sistema inteligente de control de tráfico con software de código abierto en sistemas embebidos para obtener el grado de*.

Lazo, W. A. A. (2019). *ESPECTROSCOPIA CON INFRARROJO Y TECNICAS DE MACHINE LEARNING Y DEEP LEARNING PARA LA DETECCIÓN Y CLASIFICACIÓN DE FRUTAS PARA LA AGROINDUSTRIA. CASO: ARÁNDANOS - EMPRESA TalSA -2018 "* [Universidad Privada Antenor Orrego]. http://www.gonzalezcabeza.com/documentos/CRECIMIENTO_MICROBIANO.pdf

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recogition. *Proc. OF THE IEEE*, 50.

Nelson, J., & Solawetz, J. (2021). *YOLOv5 is Here : State-of-the-Art Object Detection at 140 FPS*. https://blog.roboflow.com/yolov5-is-here/

Nielsen, M. (2019). *Neural Networks and Deep learning*. neuralnetworksanddeeplearning.com/chap6.html#introducing_convolutional_networks

Ordoñes, E. R. (2020). *DEEP LEARNING PARA LA VISIÓN ARTIFICIAL E IDENTIFICACIÓN DEL PERSONAL ADMINISTRATIVO Y DOCENTE DE LA UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC 2018 PRESENTADA*. Universidad Nacional del Altiplano.

Ponce, P. C. (2010). Inteligencia Artificial con aplicaciones a la ingeniería. In *Alfaomega, México* (Primera ed). http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Inteligencia+artificial+con+aplicaciones+a+la+ingenier&#65533;a#0

Por, P., & Aproximación, I. (2019). *Programación por imitación. aproximación al aprendizaje en ingeniería de sistemas*. *12*(2).

PROINVERSION. (2018). *Notas de Prensa / 623 MIL PERUANOS DE JUNÍN, PUNO, MOQUEGUA Y TACNA TENDRÁN INTERNET DE ALTA VELOCIDAD*. https://www.proinversion.gob.pe/modulos/NOT/NOT_DetallarNoticia.aspx?ARE=0&PFL=1&NOT=4165&month=4&year=2018

Raschka, S., & Mirjalili, V. (2019). *PYTHON MAHINE LEARNING* (2da ed.). https://yolibrospdf.com/programacion.html

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*.

Rodríguez, J. (2018). *Aplicación de técnicas de machine learning a la detección de*

ataques. 52. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81126/11/jmrodriguez85TFM0618memoria.pdf

Ruder, S. (2017). *Transfer Learning - Machine Learning ' s Next Frontier What is Transfer Learning ?* https://ruder.io/transfer-learning/

Sanlam, G. I. S. (2018). Introducción a la Inteligencia Artificial. *Intelligence*. http://turing.iimas.unam.mx/~luis/cursos/IA09/slides/s1_inteligencia_artificial.pdf

Simonyan, K., & Zisserman, A. (2015). *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION Karen*. 1–14.

Szegedy, C., Reed, S., Sermanet, P., Vanhoucke, V., & Rabinovich, A. (2014). *Going deeper with convolutions*. 1–12.

Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 10778–10787. https://doi.org/10.1109/CVPR42600.2020.01079

Torra, V. (2020). La inteligencia artificial. *La Inteligencia Artificial*, *18*(1), 58–88. https://doi.org/10.7195/ri14.v18i1.1434

Viera, G. maza. (2017). *PROCESAMIENTO DE IMÁGENES USANDO OPENCV APLICADO EN RASPBERRY PI PARA LA CLASIFICACIÓN DEL CACAO*. UNIVERSIDAD DE PIURA.

Visa, S., Ralescu, A., Ramsay, B., & Knaap, E. van der. (2011). Confusion Matrix-

based Feature Selection Sofia Visa. *ConfusionMatrix-Based Feature Selection Sofia*, *710*(January), 8.

Vizcaya, R., Martin, J., Albino, F., & Lazcano-Salas, S. (2017). *Desempeño de una red neuronal convolucional para clasificación de señales de tránsito*. *October*. https://www.researchgate.net/publication/323456954

Yegulalp, S. (2021). *What is Python ? Powerful , intuitive programming Python ' s key advantages*. 1–11. https://www.infoworld.com/article/3204016/what-is-python-powerful-intuitive-programming.htmL

ANEXOS

ANEXO A. CÓDIGO FUENTE DE YOLO V5

```
Automatically generated by Colaboratory.
Original file is located at
    https://colab.research.google.com/drive/1I62cpQez8pgeKKGmtRPXxIxFSKLcp_4U
### activar GPU
"""
!apt update
!uname -m && cat /etc/*release
!gcc --version
!uname-r
!nvidia-smi
#conectar con drive
from google.colab import drive
drive.mount('/content/drive')
#lista de archivos que se tiene en drive
!ls drive/My\ Drive
#clonando repositrio de pjreddie para preparar descarga de arquitectura darknet
# %rm -r darknet
!git clone https://github.com/pjreddie/darknet
# %cd darknet/
!pwd
# Clonando repositorio de darknet con arquitectura yoloV4
# Commented out IPython magic to ensure Python compatibility.
#%rm -r darknet
!git clone https://github.com/AlexeyAB/darknet/
# %cd darknet/
#instalando librerias
!apt install libopencv-dev python-opencv ffmpeg
# Creando archivo obj.names de los objetos a identificar.
all_classes = """Amortiguador
Cruzeta
Etiqueta
P.Doble
P.Simple
Retenida
Suspension
"""
file = """text_file = open("build/darknet/x64/data/obj.names", "w");text_file.write(all_classes);text_file.close()"""
exec(file)
```

```
# %pycat build/darknet/x64/data/obj.names
!pwd
!pwd
# Creando archivo obj.data que contine la ruta de las carpetas de imagenes de entrenamento y test
obj_data = """classes= 7
train  = build/darknet/x64/data/train.txt
valid  = build/darknet/x64/data/valid.txt
names = build/darknet/x64/data/obj.names
backup = build/darknet/x64/backup/
"""
file = """text_file = open("build/darknet/x64/data/obj.data", "w");text_file.write(obj_data);text_file.close()"""
exec(file)
# %pycat build/darknet/x64/data/obj.data
!pwd
#Dividiendo archivos de entrenamiento y validacion
import os, fnmatch
import numpy as np
train_file = open("build/darknet/x64/data/train.txt", "w")
valid_file = open("build/darknet/x64/data/valid.txt", "w")
listOfFiles = os.listdir('build/darknet/x64/data/obj/')
pattern = "*.jpg"
for f_name in listOfFiles:
  if fnmatch.fnmatch(f_name, pattern):
    if np.random.rand(1) < 0.8:
      train_file.write("build/darknet/x64/data/obj/"+f_name+"\n")
      #print ("data/obj/"+f_name)
    else:
      valid_file.write("build/darknet/x64/data/obj/"+f_name+"\n")
train_file.close()
valid_file.close()
#Count number of files
!wc -l build/darknet/x64/data/train.txt
!wc -l build/darknet/x64/data/valid.txt
"""### Mostrando datos"""
# %pycat build/darknet/x64/data/valid.txt
!pwd
# Commented out IPython magic to ensure Python compatibility.
# %cd ../..
"""descargar pesos """
```

```
# upload pretrained convolutional layer weights
#!wget http://pjreddie.com/media/files/darknet53.conv.74 para yolo v3
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137    #para yolo v4
!pwd

#Copiar los pesos a la carpeta adecuada de entrenamiento del modelo propio
# Commented out IPython magic to ensure Python compatibility.
#copy downloaded pre-trained weights (e.g. yolov4.conv.137)
# %cp ../../yolov4.conv.137 build/darknet/x64/
#%ls build/darknet/x64/
"""## copiando configuracion realizada o reescribiendo la configuracion por defecto que viene"""
# Commented out IPython magic to ensure Python compatibility.
# %cp '../../yolov4_custom2.cfg' cfg/
"""# Write Custom Training Config for YOLOv4"""
#we build config dynamically based on number of classes
#we build iteratively from base config files. This is the same file shape as cfg/yolo-obj.cfg
def file_len(fname):
  with open(fname) as f:
    for i, l in enumerate(f):
      pass
  return i + 1
num_classes = file_len('train/_darknet.labels')
print("writing config for a custom YOLOv4 detector detecting number of classes: " + str(num_classes))
#Instructions from the darknet repo
#change line max_batches to (classes*2000 but not less than number of training images, and not less than 6000), f.e. max_batches=6000 if you train for 3 classes
#change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
if    os.path.exists('./cfg/custom-yolov4-detector.cfg'):    os.remove('./cfg/custom-yolov4-detector.cfg')


with open('./cfg/custom-yolov4-detector.cfg', 'a') as f:
  f.write('[net]' + '\n')
  f.write('batch=64' + '\n')
  #####smaller subdivisions help the GPU run faster. 12 is optimal, but you might need to change to 24,36,64####
  f.write('subdivisions=24' + '\n')
  f.write('width=416' + '\n')
```

```
  f.write('height=416' + '\n')
  f.write('channels=3' + '\n')
  f.write('momentum=0.949' + '\n')
  f.write('decay=0.0005' + '\n')
  f.write('angle=0' + '\n')
  f.write('saturation = 1.5' + '\n')
  f.write('exposure = 1.5' + '\n')
  f.write('hue = .1' + '\n')
  f.write('\n')
  f.write('learning_rate=0.001' + '\n')
  f.write('burn_in=1000' + '\n')
  ######you can adjust up and down to change training time#####
  ##Darknet does iterations with batches, not epochs####
  max_batches = num_classes*2000
  #max_batches = 2000
  f.write('max_batches=' + str(max_batches) + '\n')
  f.write('policy=steps' + '\n')
  steps1 = .8 * max_batches
  steps2 = .9 * max_batches
  f.write('steps='+str(steps1)+','+str(steps2) + '\n')
#Instructions from the darknet repo
#change line classes=80 to your number of objects in each of 3 [yolo]-layers:
#change [filters=255] to filters=(classes + 5)x3 in the 3 [convolutional] before each [yolo] layer, keep in mind that it only has to be the last [convolutional] before each of the [yolo] layers.
  with open('cfg/yolov4-custom2.cfg', 'r') as f2:
    content = f2.readlines()
    for line in content:
      f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 0,1,2' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')
  with open('cfg/yolov4-custom3.cfg', 'r') as f3:
    content = f3.readlines()
```

```
    for line in content:
      f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 3,4,5' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243,
459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')
  with open('cfg/yolov4-custom4.cfg', 'r') as f4:
    content = f4.readlines()
    for line in content:
      f.write(line)
    num_filters = (num_classes + 5) * 3
    f.write('filters='+str(num_filters) + '\n')
    f.write('activation=linear')
    f.write('\n')
    f.write('\n')
    f.write('[yolo]' + '\n')
    f.write('mask = 6,7,8' + '\n')
    f.write('anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243,
459, 401' + '\n')
    f.write('classes=' + str(num_classes) + '\n')

  with open('cfg/yolov4-custom5.cfg', 'r') as f5:
    content = f5.readlines()
    for line in content:
      f.write(line)
print("file is written!")

#like the number of subdivisions 64 runs faster but Colab GPU may not be big enough
#if Colab GPU memory is too small, you will need to adjust subdivisions to 16
# %cat cfg/custom-yolov4-detector.cfg
!pwd
#cambiando paramtros de cpñab antes de entrenamiento
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/g' Makefile
```

```
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
#ejecutar archivo make
# %pycat Makefile  #error vrificsr luego
!make
"""## MODELO ENTRENAR"""
!pwd
# %cd /content/drive/My Drive/deteccion/darknet/darknet
# need to set our custom cfg to test mode
# %cd 'cfg/'
!sed -i 's/batch=1/batch=64/' yolov4_custom2.cfg
!sed -i 's/subdivisions=1/subdivisions=32/' yolov4_custom2.cfg
# %cd ..
#inicio de entrenamiento configurando ubicacion de archivos que requiere
!./darknet detector train build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4.conv.137 -dont_show -map
# define helper functions
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
#   %matplotlib inline
  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()
# use this to upload files
def upload():
  from google.colab import files
  uploaded = files.upload()
  for name, data in uploaded.items():
    with open(name, 'wb') as f:
      f.write(data)
      print ('saved file', name)
# use this to download a file
def download(path):
  from google.colab import files
  files.download(path)
```

```
# show chart.png of how custom object detector did with training
imShow('chart.png')
# muestra los pesos que se haya entrenado
# %ls build/darknet/x64/backup/*.weights
# Commented out IPython magic to ensure Python compatibility.
# rename pre-trained weights file
# %cp -r build/darknet/x64/backup/yolov4_custom2_last.weights build/darknet/x64/yolov4_custom2_2000.weights
# Commented out IPython magic to ensure Python compatibility.
# %ls build/darknet/x64/backup/*.weights
!./darknet detector train build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_2000.weights -dont_show -map
"""verifcar el 7020: 0.9209   eso deve ser menor que 1 para que no ahya mucha perdida
# show chart.png of how custom object detector did with training
imShow('chart.png')
!pwd

"""### correindo despues de 7000 iteraciones ya que se paralizo en dicha epoca por uso de plataforma colab de 12 horas"""
!./darknet detector train build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_7000.weights -dont_show -map

# show chart.png of how custom object detector did with training
imShow('chart.png')
# Commented out IPython magic to ensure Python compatibility.
# rename pre-trained weights file
# %cp -r build/darknet/x64/backup/yolov4_custom2_last.weights build/darknet/x64/yolov4_custom2_9700.weights
### corriendo despues de 9700 iteraciones
!./darknet detector train build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_9700.weights -dont_show -map
# Commented out IPython magic to ensure Python compatibility.
# rename pre-trained weights file
# %cp -r chart.png ../../chart2.png
# show chart.png of how custom object detector did with training
imShow('chart.png')
# Commented out IPython magic to ensure Python compatibility.
# rename pre-trained weights file
# %cp -r build/darknet/x64/backup/yolov4_custom2_last.weights build/darknet/x64/yolov4_custom2_11300.weights
#entrenando desde 11300
```

!./darknet detector train build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_11300.weights -dont_show -map

show chart.png of how custom object detector did with training

imShow('chart.png')

"""# Step 6: Checking the Mean Average Precision (mAP) of Your Model

If you didn't run the training with the '-map- flag added then you can still find out the mAP of your model after training. Run the following command on any of the saved weights from the training to see the mAP value for that specific weight's file. I would suggest to run it on multiple of the saved weights to compare and find the weights with the highest mAP as that is the most accurate one!

NOTE: If you think your final weights file has overfitted then it is important to run these mAP commands to see if one of the previously saved weights is a more accurate model for your classes

"""

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_2000.weights

Commented out IPython magic to ensure Python compatibility.

%ls build/darknet/x64/*.weights

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_7000.weights

"""# verificando MAPpara 9700 iteraciones"""

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_9700.weights

"""### verificnado MAP para 11300 iteraciones"""

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_11300.weights

"""## Verificando MAP a 14000 iteraciones"""

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights

"""# Map para el best"""

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/backup/yolov4_custom2_best.weights

! ./ darknet detector map data / obj.data path_to_weights.weights -dont_show -ext_output <path_to_the_txt_that_has_all_the_images_you_want_to_detect.txt> result.txt

!./darknet detector map build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/backup/yolov4_custom2_best.weights -dont_show -ext_output <build/darknet/x64/data/valid.txt> result.txt

download ('result.txt')

"""# Step 7: Run Your Custom Object Detector!!!

You have done it! You now have a custom object detector to make your very own detections. Time to test it out and have some fun!

```
"""
!pwd
# Commented out IPython magic to ensure Python compatibility.
# need to set our custom cfg to test mode
!sed -i 's/batch=64/batch=1/' yolov4_custom2.cfg
!sed -i 's/subdivisions=32/subdivisions=1/' yolov4_custom2.cfg
# %cd ..
# Commented out IPython magic to ensure Python compatibility.
!pwd
# %cd cfg/
# Commented out IPython magic to ensure Python compatibility.
# %pycat yolov4_custom2.cfg
# Commented out IPython magic to ensure Python compatibility.
!pwd
# %cd darknet/
# run your custom detector with this command (upload an image to your google drive to test, thresh flag sets accuracy that detection must be in order to show it)
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights VALIDACION/3.jpg -thresh 0.5
imShow('predictions.jpg')
"""## Para guardar los Puede generar las coordenadas del cuadro delimitador para cada detección con el indicador '-ext_output'. Esta bandera de salidas externas le dará algunos detalles """
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output ../../VALIDACION/9.jpg -dont_show -out result1.json
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -ext_output
imShow('predictions.jpg')
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output ../../VALIDACION/16.jpg -dont_show -out result1.json
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -ext_output
imShow('predictions.jpg')
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output ../../VALIDACION/18.jpg -dont_show -out result1.json
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -ext_output
```

```
imShow('predictions.jpg')
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output ../../VALIDACION/20.jpg -dont_show -out result1.json
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -ext_output
imShow('predictions.jpg')
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output ../../VALIDACION/12.jpg -dont_show -out result10.json
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -ext_output
imShow('predictions.jpg')
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output VALIDACION/2.jpg -dont_show -out result10.json
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -ext_output
imShow('predictions.jpg')
download('result10.json')
"""# Step 9: Multiple Images at Once
YOLOv4 object detections can be run on multiple images at once. This is done through having a text file which has the paths to several images that you want to have the detector run on.
The .txt file should be in this format. One path to an image per line.
This file is stored to my Google Drive root directory and holds the path to three images within my Google Drive images folder.
## Save Results to .JSON File
Here is an example of saving the multiple image detections to a .JSON file.
"""
!pwd
# this creates a symbolic link so that now the path /content/gdrive/My\ Drive/ is equal to /mydrive
!ln -s /content/drive/My\ Drive/ /mydrive
!ls /mydrive
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -ext_output -dont_show -out result11.json < build/darknet/x64/data/valid.txt
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext_output -dont_show -out result.json < /mydrive/images.txt
```

```
download('result.json')
"""## Saving Results to a .txt File
You can also save the results of running multiple images to a text file.
"""
#!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show -ext_output < /mydrive/images.txt > result.txt
!./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_custom2.cfg build/darknet/x64/yolov4_custom2_14000.weights -dont_show -ext_output < validacion2.txt> result.txt
#-thresh 0.5
#imShow('predictions.jpg')
!pwd
# Commented out IPython magic to ensure Python compatibility.
# %ls VALIDACION/
download('result.txt')
```

ANEXO B. CÓDIGO DE YOLOV5 PYTORCH

```
# -*- coding: utf-8 -*-
#yolov5postes01.ipynb
#Automatically generated by Colaboratory.
#Original file is located at
 #   https://colab.research.google.com/drive/1vHmdNlLglqdqA1GSmyLlaK9lmi4ao0Eu
#conectando con drive
from google.colab import drive
drive.mount('/content/drive')
# %cd drive/MyDrive/deteccion/Postesv5/
# %cd drive/MyDrive/deteccion/Postesv5/
!git clone https://github.com/ultralytics/yolov5  # clone repo
# %cd yolov5
# %pip install -qr requirements.txt  # install dependencies
import torch
from IPython.display import Image, clear_output  # to display images
clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
import torch
from IPython.display import Image, clear_output  # to display images
clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
!gdown --id 1ZycPS5Ft_0vlfgHnLsfvZPhcH6qOAqBO -O data/clothing.yaml
!gdown --id 1czESPsKbOWZF7_PkCcvRfTiUUJfpx12i -O models/yolov5x.yaml
# %cd /content/drive/My Drive/deteccion/Postesv5/yolov5
# %cp data/clothing.yaml data/postes.yaml
# %cat data/postes.yaml
#Posterior a ello se editó en editor de textos sublime u otro"""
## copiando modelos y editando en editores de texto
# %cp models/yolov5x.yaml models/yolov5xpostes.yaml
"""### Acoplando según encarpetado"""
original_data_dir = '/content/drive/MyDrive/deteccion/data_aument_yolo'
import glob
# ファイル数カウント(Count the number of files)
```

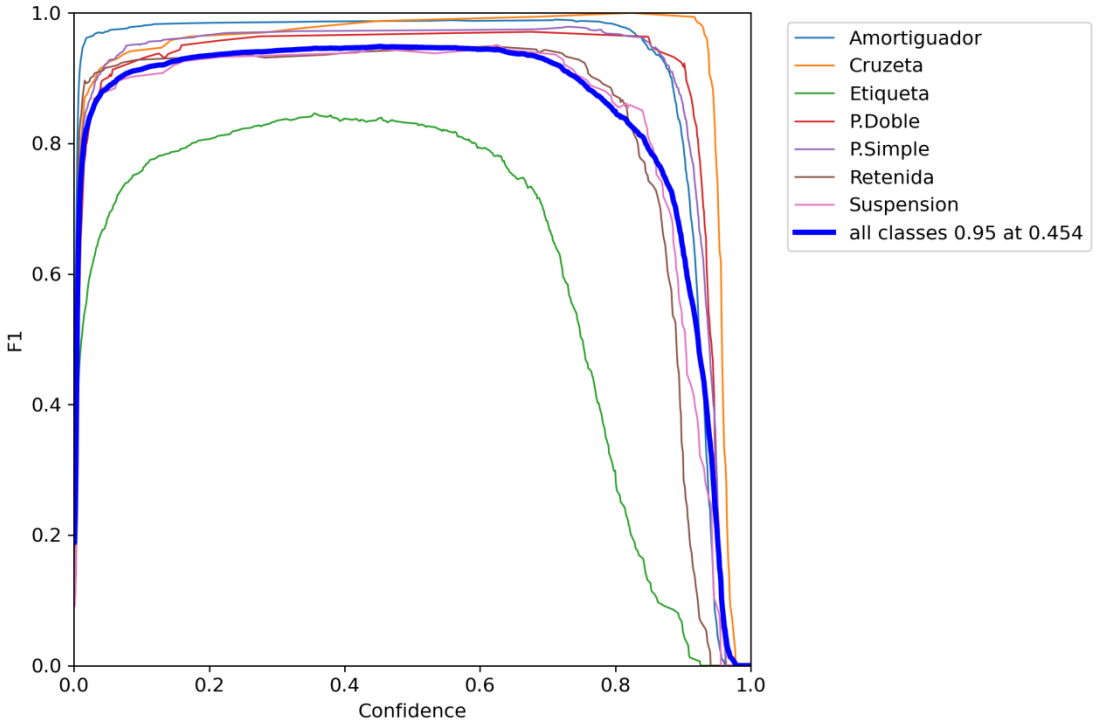
```
file_count = len(glob.glob(original_data_dir + '/*.jpg'))
print('File count : ' + str(file_count))
archi1=open("/content/drive/MyDrive/deteccion/Postesv5/postesData/train.txt","r")
lineas=archi1.readlines()
entrenamiento=lineas
print('El archivo tiene', len(lineas), 'líneas')
print('El contenido del archivo')
for linea in lineas:
    print(linea, end='')
archi1.close()
len(entrenamiento)
# Commented out IPython magic to ensure Python compatibility.
import os.path as path
global str
train_image='/content/drive/MyDrive/deteccion/Postesv5/postesData/imagenes/entrenamiento'
train_labels='/content/drive/MyDrive/deteccion/Postesv5/postesData/labels/entrenamiento'
#file_list = glob.glob(original_data_dir + '/*.jpg')
sample_list = entrenamiento
concat='/content/drive/MyDrive/deteccion/darknet/darknet/'
direc='build/darknet/x64/data/obj'
count=0
for index, filepath in enumerate(sample_list):
  filepath=filepath.replace(direc,train_image)
  filepath=filepath.strip('\n')
  if path.exists(filepath):
    filepath1=filepath.replace('imagenes','labels')
    filepath1=filepath1.replace('jpg','txt')
    print(filepath)
#     %rm filepath
   # %rm filepath1
print(count)
"""## copiando data a validaciondes"""
archi1=open("/content/drive/MyDrive/deteccion/Postesv5/postesData/valid.txt","r")
lineas=archi1.readlines()
entrenamiento=lineas
```

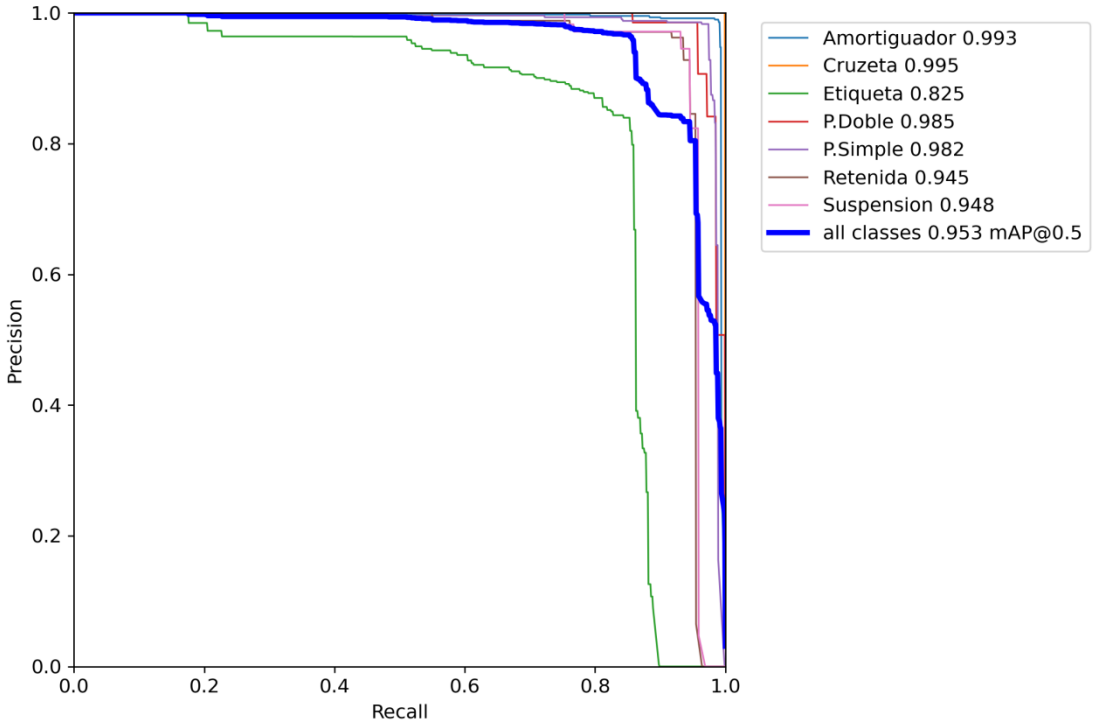
```
print('El archivo tiene', len(lineas), 'líneas')
print('El contenido del archivo')
#for linea in lineas:
 #   print(linea, end='')
#archi1.close()
test_image='/content/drive/MyDrive/deteccion/Postesv5/postesData/imagenes/validacion'
test_labels='/content/drive/MyDrive/deteccion/Postesv5/postesData/labels/validacion'
file_list = glob.glob(original_data_dir + '/*.jpg')
sample_list = entrenamiento
concat='/content/drive/MyDrive/deteccion/darknet/darknet/'
for index, filepath in enumerate(sample_list):
  buscar='jpg\n'
  reemplazar1='jpg'
  reemplazar2="txt"
  filepath1=(filepath.replace(buscar,reemplazar1))
  filepath2=(filepath.replace(buscar,reemplazar2))
  shutil.copy2(concat+filepath1, test_image)
  shutil.copy2(concat+filepath2, test_labels)
"""# verificando cantidades de datos en entrenamiento y test"""
file_list_train = glob.glob('/content/drive/MyDrive/deteccion/Postesv5/postesData/images/entrenamiento' + '/*.jpg')
print('ENTRENAMIENTO: ',len(file_list_train))
file_list_value = glob.glob('/content/drive/MyDrive/deteccion/Postesv5/postesData/images/validacion' + '/*.jpg')
print('TEST: ',len(file_list_value))
"""# codigo de entrenamiento"""
#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic
@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
# # temp fix for image_weights=opt.image_weights error, see: https://github.com/ultralytics/yolov5/issues/1550
# %%writetemplate drive/MyDrive/deteccion/Postesv5/yolov5/train.py
```

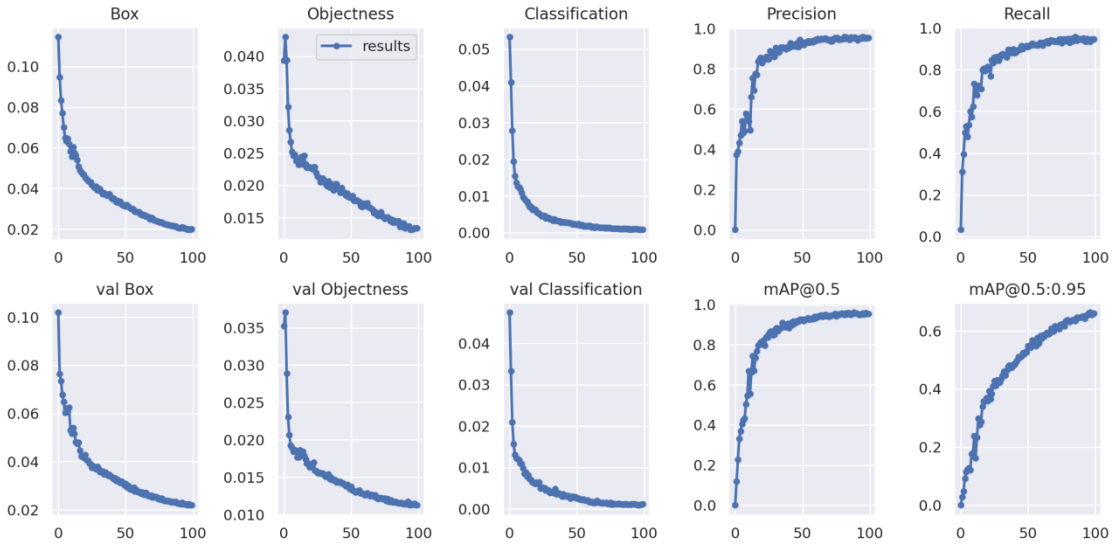
```
"""#TRAININIG"""
!pwd
# Tensorboard (optional)
# %load_ext tensorboard
# %tensorboard --logdir runs/train
# Weights & Biases (optional)
# %pip install -q wandb
!wandb login  # use 'wandb disabled' or 'wandb enabled' to disable or enable


!git clone https://github.com/ultralytics/yolov5  # clone repo
# %cd yolov5
# %pip install -qr requirements.txt  # install dependencies
# Train YOLOv5x for 100 epochs
!python train.py --img 640 --batch 16 --epochs 100 --data postes.yaml --cfg yolov5xx.yaml --weights yolov5x.pt --nosave --cache
import torch
from IPython.display import Image, clear_output  # to display images
clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
# %cp data/coco128.yaml data/postes.yaml
# %cp models/yolov5x.yaml models/yolov5xx.yaml
#Todos los resultados del entrenamiento se guardan en `runs / train /` con directorios de ejecución incrementales, es decir, `runs / train / exp2`,` runs / train / exp3` etc.
"""

"""#Las pérdidas de entrenamiento y las métricas de rendimiento también se registran en Tensorboard y en un archivo de registro results.txt personalizado que se representa como results.png (a continuación) después de que se completa el entrenamiento. Aquí mostramos YOLOv5s entrenados en COCO128 hasta 300 épocas, comenzando desde cero (azul) y desde preentrenados --weights yolov5s.pt (naranja)"""
from utils.plots import plot_results
plot_results(save_dir='runs/train/exp2')  # plot all results*.txt as results.png
Image(filename='runs/train/exp2/results.png', width=800)
#
"""detect.py runs inference on a variety of sources, downloading models automatically from the latest YOLOv5 release."""
# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/deteccion/Postesv5/yolov5
import torch
```

```
from IPython.display import Image, clear_output  # to display images
!python detect.py --weights runs/train/exp2/weights/last.pt --img 640 --conf 0.25 --source /content/drive/MyDrive/deteccion/VALIDACION/
#Image(filename='runs/detect/exp2/1.jpg', width=600)
d=[1,2,3,4,5,6,7]
for i in d:
  Image(filename=runs/detect/exp5/$i.jpg, width=600)
for i in 7
```

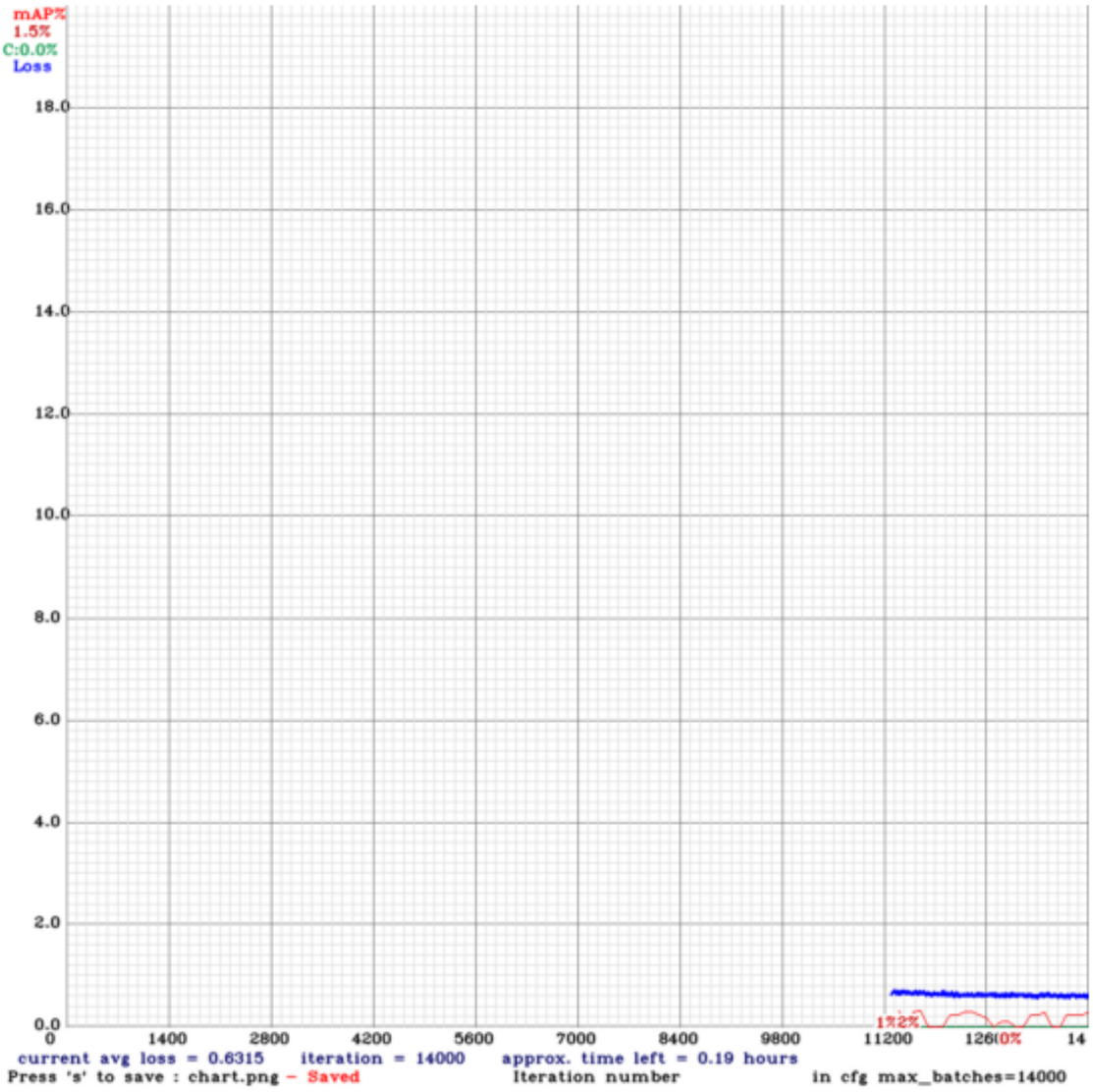
ANEXO C. RESULTADOS CON REPORTE DE PYTORCH

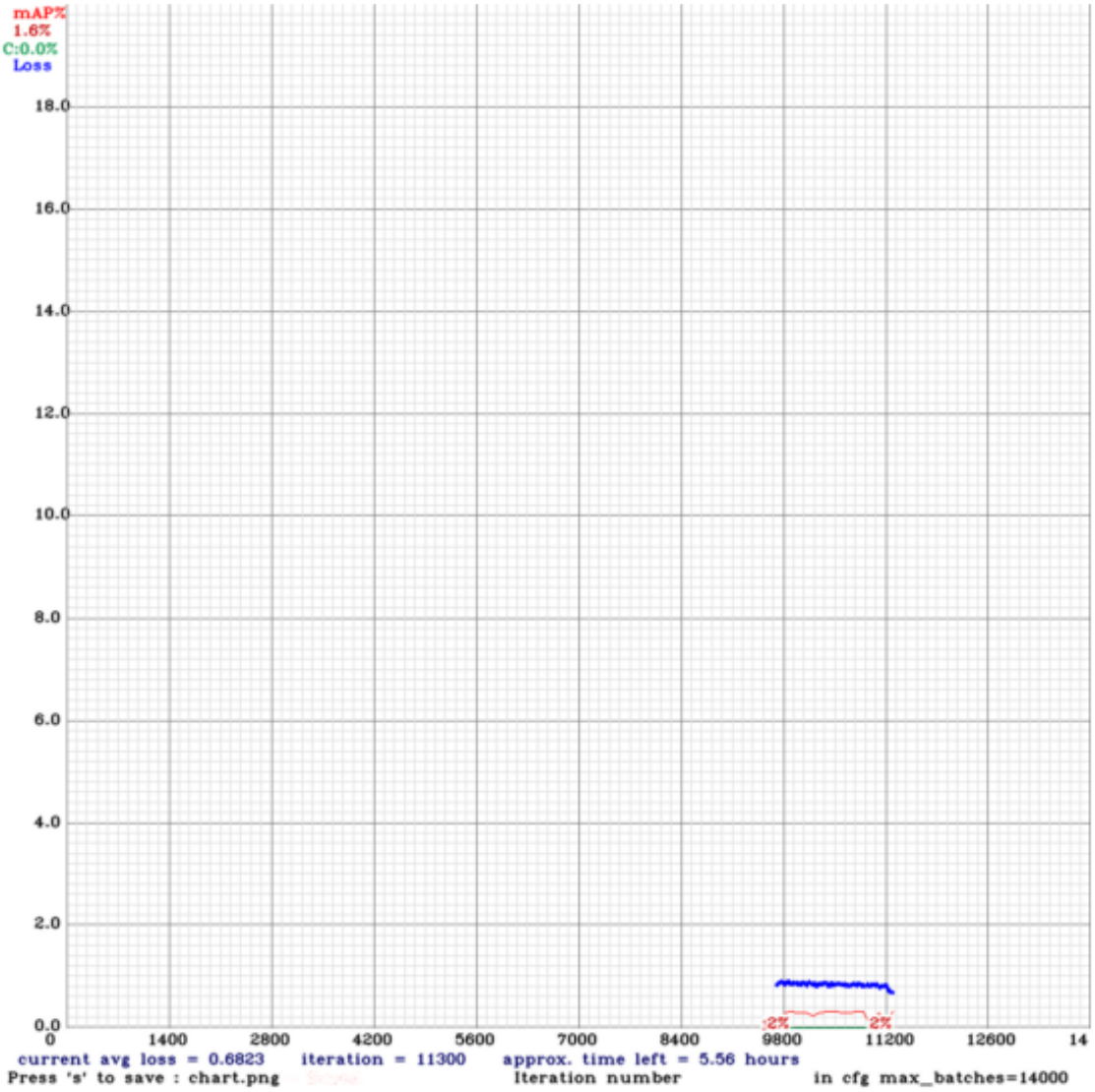


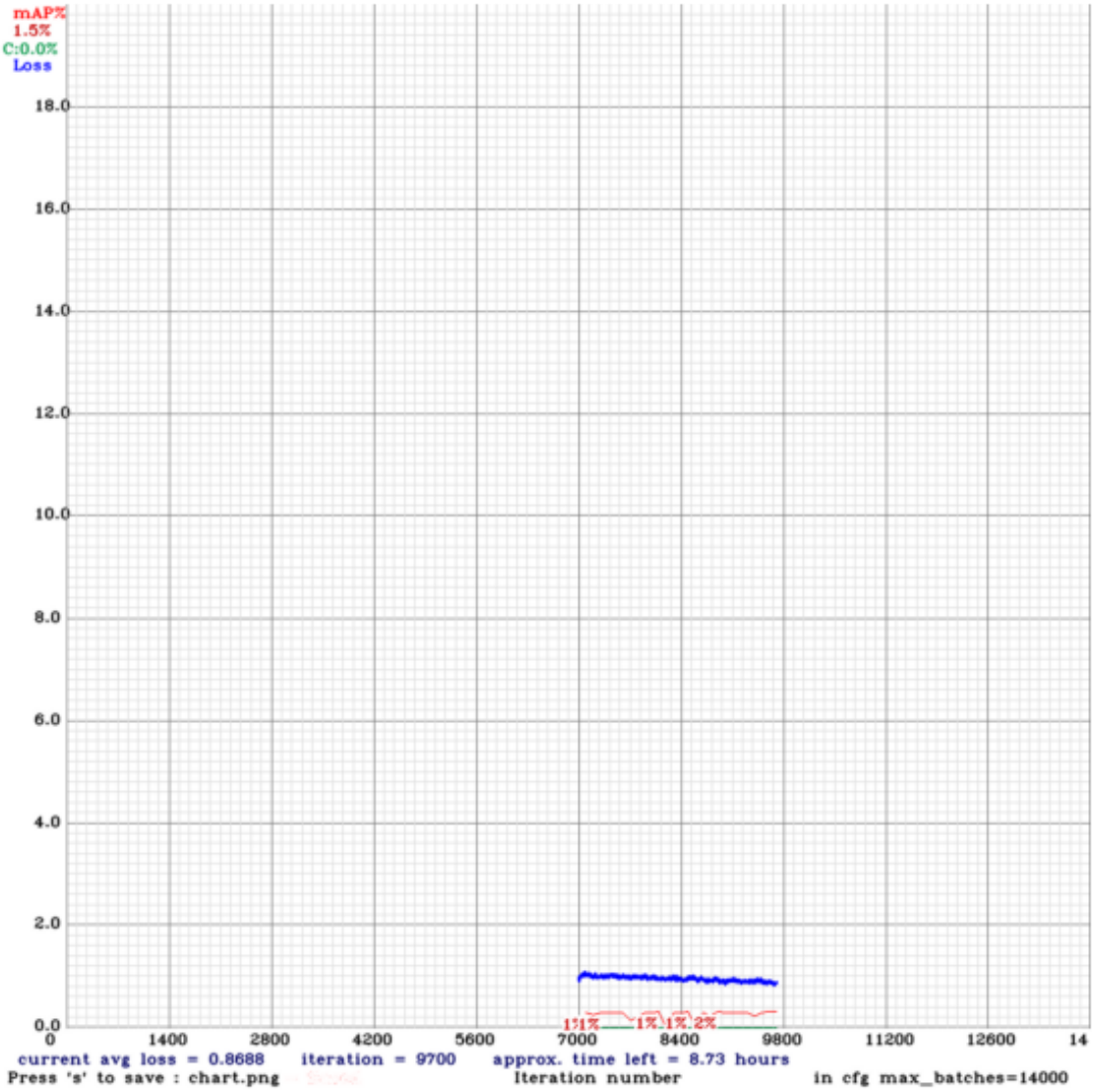


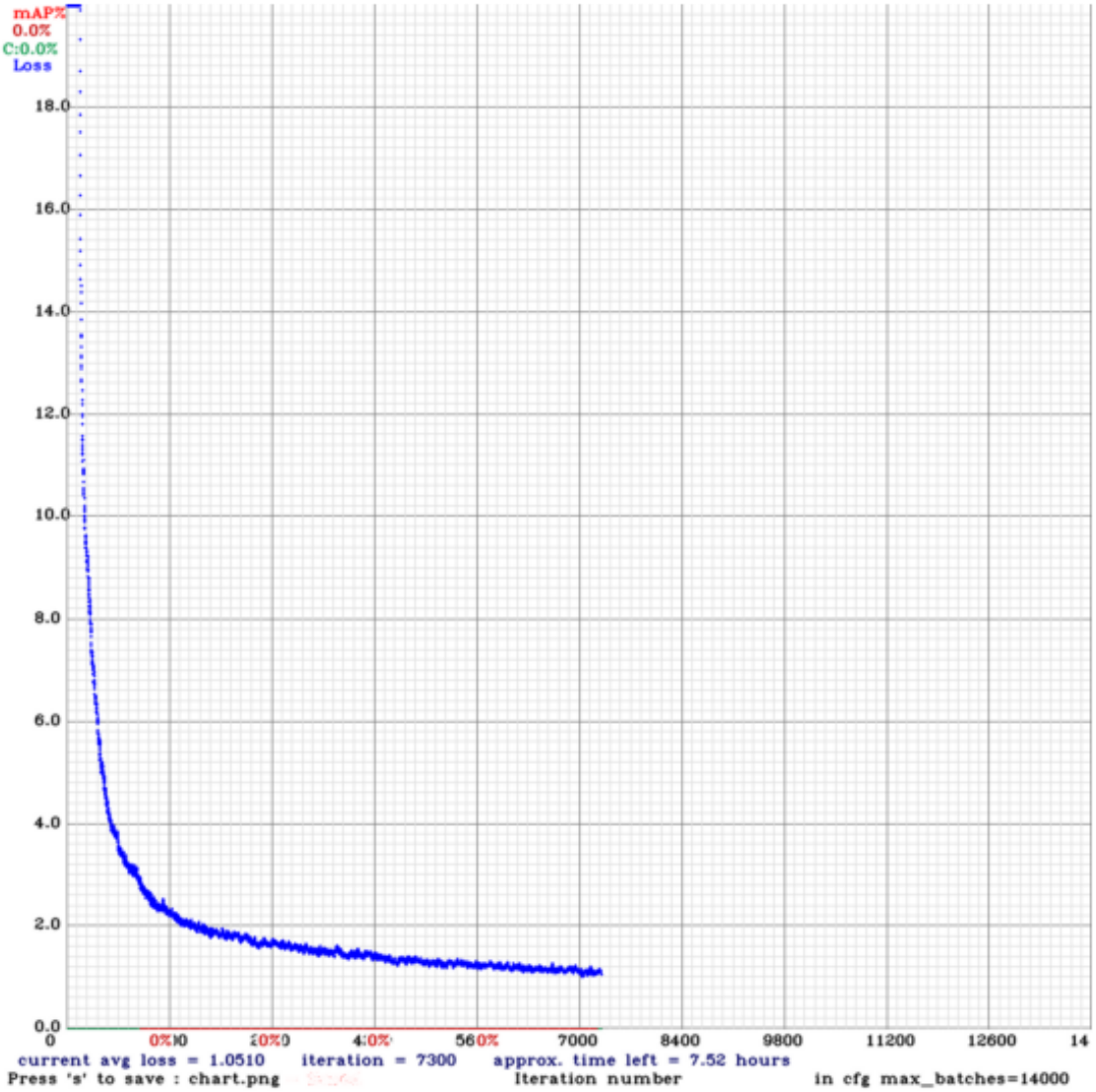


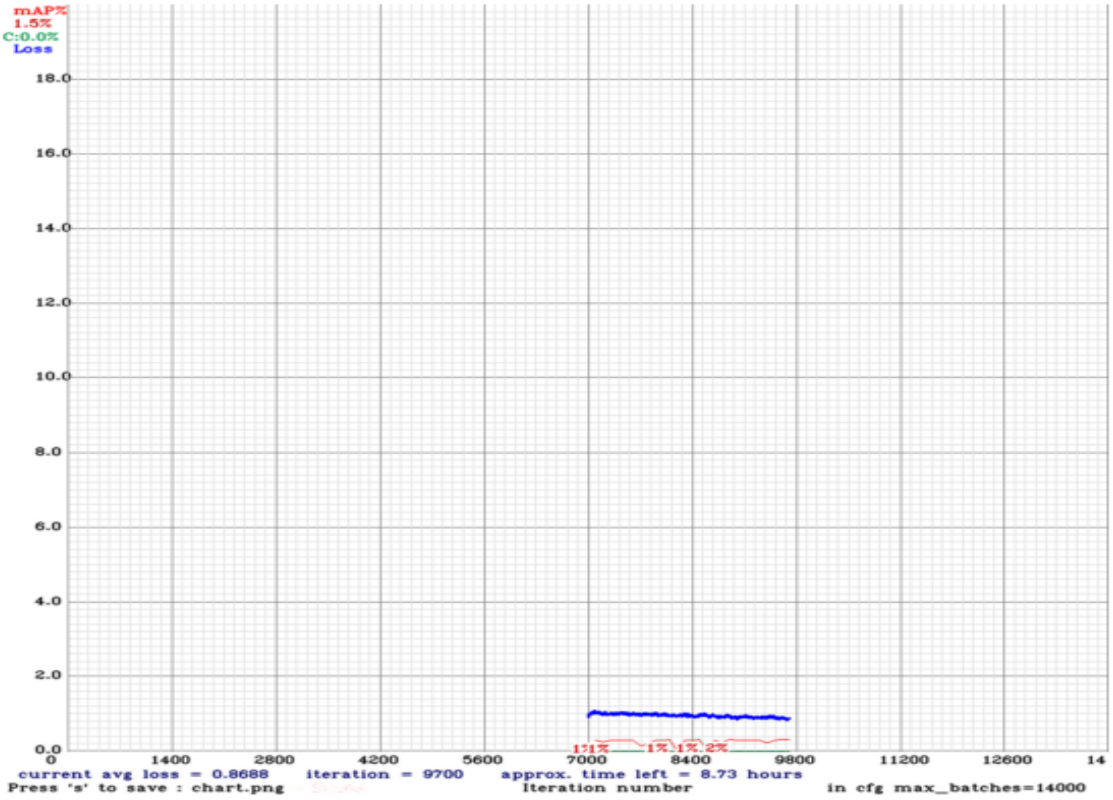
ANEXO D. GRÁFICO DE PÉRDIDA EN DIFERENTES ÉPOCAS DE ENTRENAMIENTO DE YOLOV4



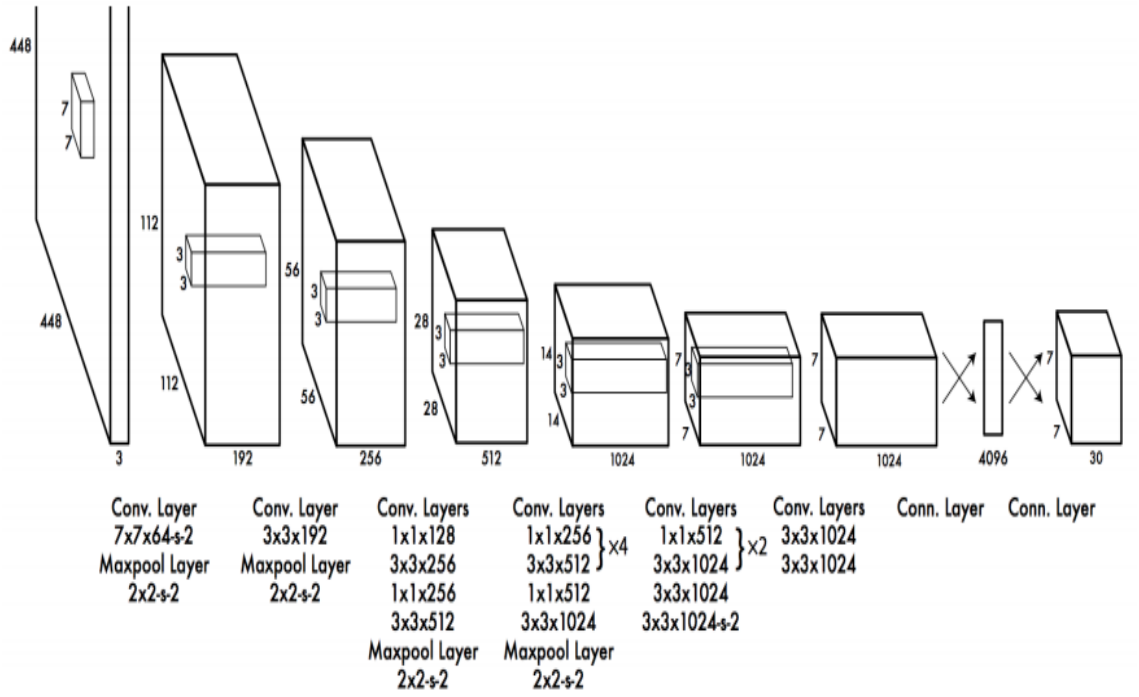








ANEXO E. ARQUITECTURA YOLO



ANEXO F. CONFIGURACIÓN DE ARCHIVO YOLO V5.YAML

```
# parameters
nc: 7  # number of classes
depth_multiple: 1.33  # model depth multiple
width_multiple: 1.25  # layer channel multiple

# anchors
anchors:
  - [116,90, 156,198, 373,326]  # P5/32
  - [30,61, 62,45, 59,119]  # P4/16
  - [10,13, 16,30, 33,23]  # P3/8

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
```

```
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
  ]

# YOLOv5 head
head:
  [[-1, 3, BottleneckCSP, [1024, False]],  # 9

   [-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],
   [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1]],  # 18 (P3/8-small)

   [-2, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],
   [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1]],  # 22 (P4/16-medium)

   [-2, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],
   [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1]],  # 26 (P5/32-large)

   [[], 1, Detect, [nc, anchors]],  # Detect(P5, P4, P3)
  ]
```

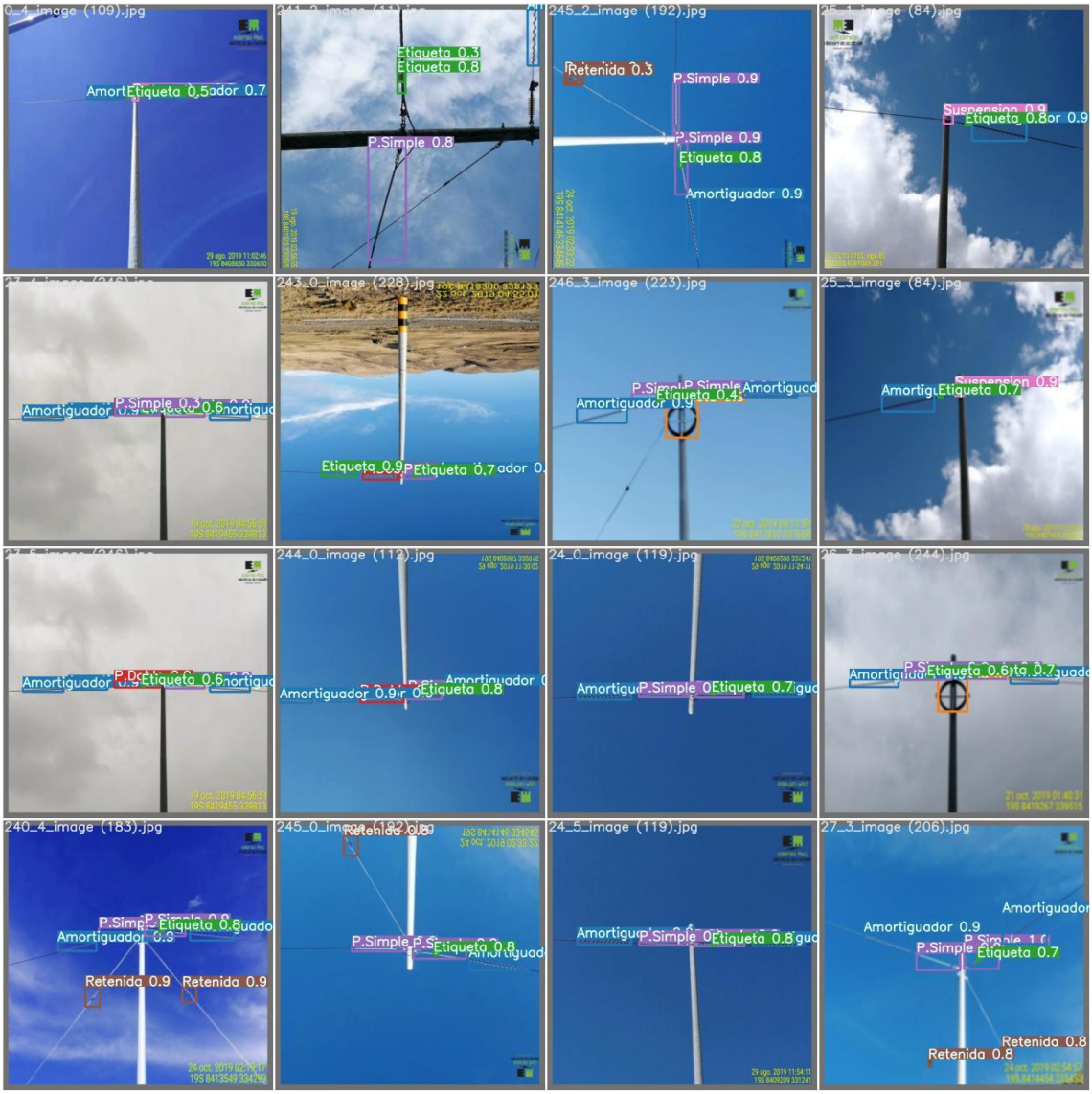
ANEXO G: FORMATO DE ANOTACIÓN PASCALVOC .XML CON LLABELLING

<annotation>
<folder>IMAGENES POSTES</folder>
<filename>41.jpg</filename>
<path>C:\Users\Wilber\Desktop\IMAGENES POSTES\41.jpg</path>
<source>
<database>Unknown</database>
</source>
<size>
<width>5120</width>
<height>3840</height>
<depth>3</depth>
</size>
<segmented>0</segmented>
<object>
<name>Amortiguador</name>
<pose>Unspecified</pose>
<truncated>1</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>4100</xmin>
<ymin>1629</ymin>
<xmax>5120</xmax>
<ymax>1755</ymax>
</bndbox>
</object>
<object>
<name>Amortiguador</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>

<xmin>3163</xmin>
<ymin>1704</ymin>
<xmax>4025</xmax>
<ymax>1816</ymax>
</bndbox>
</object>
<object>
<name>Amortiguador</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>150</xmin>
<ymin>2138</ymin>
<xmax>554</xmax>
<ymax>2246</ymax>
</bndbox>
</object>
<object>
<name>Amortiguador</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>608</xmin>
<ymin>2031</ymin>
<xmax>1227</xmax>
<ymax>2154</ymax>
</bndbox>
</object>
<object>
<name>P.Doble</name>
<pose>Unspecified</pose>
<truncated>0</truncated>

<difficult>0</difficult>
<bndbox>
<xmin>1320</xmin>
<ymin>1852</ymin>
<xmax>2020</xmax>
<ymax>2028</ymax>
</bndbox>
</object>
<object>
<name>P.Doble</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>2135</xmin>
<ymin>1798</ymin>
<xmax>3011</xmax>
<ymax>1893</ymax>
</bndbox>
</object>
</annotation>

ANEXO H. PREDICCIONES CON CARPETA DE VALIDACIÓN EN YOO V5



ANEXO I. CARACTERÍSTICAS GPU GOOGLE COLAB

```
wed Feb 24 19:12:35 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.39       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   56C    P8    11W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID                                                        Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

ANEXO J. RESULTADOS DE PRUEBA 6 CON YOLO V4 EN GOOGLE COLAB

Total BFLOPS 98.989
avg_outputs = 784537
Allocate additional workspace_size = 52.43 MB
Loading weights from build/darknet/x64/yolov4_custom2_14000.weights...
seen 64, trained: 896 K-images (14 Kilo-batches_64)
Done! Loaded 107 layers from weights-file
calculation mAP (mean average precision)...

Detection layer: 82 - type = 28
Detection layer: 94 - type = 28
Detection layer: 106 - type = 28
282
detections_count = 2031, unique_truth_count = 1491

class_id = 0, name = Amortiguador, ap = 93.42% (TP = 464, FP = 28)
class_id = 1, name = Cruzeta, ap = 100.00% (TP = 80, FP = 0)
class_id = 2, name = Etiqueta, ap = 63.53% (TP = 233, FP = 85)
class_id = 3, name = P.Doble, ap = 96.45% (TP = 62, FP = 7)
class_id = 4, name = P.Simple, ap = 96.29% (TP = 342, FP = 16)
class_id = 5, name = Retenida, ap = 88.22% (TP = 95, FP = 9)
class_id = 6, name = Suspension, ap = 88.24% (TP = 65, FP = 9)
for conf_thresh = 0.25, precision = 0.90, recall = 0.90, F1-score = 0.90
for conf_thresh = 0.25, TP = 1341, FP = 154, FN = 150, average IoU = 68.44 %
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.894490, or 89.45 %
Total Detection Time: 109 Seconds

ANEXO K. RESULTADOS DE MEJORES METRICAS DE YOLOV5 PYTORCH EN GOOGLE COLAB

Class	Images	Targets	P	R	mAP@.5	mAP@.5:.95: 100
all	283	1.49e+03	0.953	0.945	0.953	0.661
Amortiguador	283	490	0.983	0.992	0.993	0.719
Cruzeta	283	80	0.976	1	0.995	0.877
Etiqueta	283	313	0.87	0.812	0.825	0.315
P.Doble	283	70	0.977	0.957	0.985	0.785
P.Simple	283	356	0.971	0.975	0.982	0.746
Retenida	283	109	0.948	0.936	0.945	0.572
Suspension	283	73	0.945	0.944	0.948	0.612

Optimizer stripped from runs/train/exp2/weights/last.pt, 175.1MB

Images sizes do not match. This will causes images to be display incorrectly in the UI.

100 epochs completed in 4.20 hours.